

**TECHNOLOGY TRANSFER, ARCHITECTURAL DESIGN
AND INTEGRATED BUILDING DESIGN SYSTEMS**

Ming Sun, BSc. (Architecture)

NEWCASTLE UNIVERSITY LIBRARY

093 51100 1

Thesis L5132

August 1993

**Department of Architecture
University of Newcastle upon Tyne**

**This thesis is submitted to the Council for National Academic Awards in part
fulfilment of the requirements for the degree of Doctor of Philosophy**

ABSTRACT

This study investigates issues concerning the technology transfer from research to practice in the building industry. It is divided broadly into two parts. The first part concentrates on current problems in this transfer process. The second part explores the potential of an integrated building design system as a solution to the existing problems.

It has been widely acknowledged that many research findings and proven technologies have not been fully utilised by architects in practice. A review of the current research and design practice in the building industry has revealed a number of obstacles to this transfer process. One of the major obstacles is the gap between architects' requirements in terms of technical support and the provision of existing design support systems. Based on extensive analyses, it has been concluded that an integrated approach is needed for the provision of design supports.

The rapid increase in the application of computing facilities in design practice provides a promising platform for the development of a computer based Integrated Building Design System (IBDS). The main characteristics of an IBDS include (1) supporting multiple design tasks, (2) integrated and maintaining information exchange between different tasks, (3) responsive to architects' requests. A prototyping study is conducted in a wider context of a European research project, **Computer Modelling in Building Industry in Europe (COMBINE)**. An IBDS prototype, MultiCAD, has been developed as a result, which demonstrates the concepts and feasibility of an integrated building design system in design support. It is expected that IBDS, such as MultiCAD, will play a key role in the improvement of architectural design support and hence the technology transfer in the building industry.

Finally, conclusions of the study have been summarised and suggestions for future research in this area are outlined.

ACKNOWLEDGEMENTS

I would like to express my grateful appreciation to my supervisor, Professor John Wiltshire, for his support, patience, constructive suggestions and criticism during the conduct of this study. I also express my thanks to Mr. Steve Lockley and Mr. Dave Sedgwick for their team work and valuable help during the development of MultiCAD prototype.

Finally appreciation to every member of the Building Science Section of the Department for such an enjoyable work environment.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Introduction	1-1
1.2 Background and Objective of the Study.....	1-1
1.2.1 The context of this study	1-1
1.2.2 Current research in technology transfer	1-3
1.3 Conduct of this Study	1-4
1.3.1 Review and analysis	1-5
1.3.2 Prototype development.....	1-6
1.4 The Thesis	1-7

CHAPTER 2

TECHNOLOGY TRANSFER FROM RESEARCH TO PRACTICE

2.1 Introduction	2-1
2.2 Definitions	2-1
2.3 Building Research and Practice.....	2-6
2.3.1 The separation of research and practice	2-6
2.3.2 Building research.....	2-7
2.3.3 Building practice.....	2-8
2.4 The Need for Technology Transfer	2-9
2.5 Key Actors in the Technology Transfer Process.....	2-11
2.6 Technology Transfer Media	2-14
2.7 Influencing Factors	2-16
2.7.1 Attitudes and skills	2-16
2.7.2 Communication technology.....	2-17
2.8 Existing Barriers	2-17
2.8.1 Problems with administration.....	2-17
2.8.2 Problems with knowledge creation	2-18
2.8.3 Problems with knowledge dissemination	2-18
2.8.4 Problems with knowledge utilisation	2-19
2.9 Improving the Technology Transfer Process	2-19
2.9.1 The funding bodies	2-20
2.9.2 The research community	2-20
2.9.3 The educational institutions.....	2-21
2.9.4 The intermediary bodies	2-21
2.9.5 The design profession.....	2-21
2.10 Summary.....	2-22

CHAPTER 3

ARCHITECTURAL DESIGN AND INFORMATION REQUIREMENTS

3.1 Introduction	3-1
3.2 The Nature of Design	3-1
3.2.1 Definition of design.....	3-1
3.2.2 Design form and context.....	3-3

3.2.3 Art, science and design	3-4
3.2.4 Creativity and rationality	3-5
3.2.5 Conclusion	3-6
3.3 Design Process Models.....	3-7
3.3.1 Normal linear process	3-8
3.3.2 Iterative linear process.....	3-10
3.3.3 Cyclic iterative process.....	3-14
3.3.4 Characteristics of the design process.....	3-16
3.4 Architectural Design Practice.....	3-17
3.4.1 How do designers work?	3-18
3.4.2 How is published information used?	3-20
3.4.3 How do designers learn?.....	3-22
3.5 Information Requirements at the Early Design Stage.....	3-24
3.5.1 Domain of the early design stage	3-24
3.5.2 Designers' information requirement	3-24
3.5.3 Criteria for information provision	3-26
3.6 Summary.....	3-27

CHAPTER 4

REVIEW OF EXISTING DESIGN SUPPORT MEASURES

4.1 Introduction	4-1
4.2 Components of a Design Support System.....	4-1
4.3 Classification of Design Support Systems	4-2
4.3.1 Printed materials	4-3
4.3.2 Visual materials	4-6
4.3.3 Computer applications.....	4-7
4.3.4 Summary of current design support provision	4-9
4.4 Evaluation of Existing Design Support Systems.....	4-10
4.4.1 Validity and authority	4-11
4.4.2 Interaction and responsiveness	4-12
4.4.3 Availability and accessibility	4-13
4.4.4 Design oriented solutions	4-14
4.4.5 Concluding remarks.....	4-15
4.5 Recommended Improvements	4-15
4.6 Summary.....	4-17

CHAPTER 5

DEVELOPMENT OF COMPUTER APPLICATIONS IN ARCHITECTURE

5.1 Introduction	5-1
5.2 Application of Computers	5-1
5.2.1 The computing environment.....	5-1
5.2.2 What does the computer offer?.....	5-4
5.2.3 Computers in architecture.....	5-5
5.3 Computer Applications in Architecture	5-7
5.3.1 Computer aided design and drafting.....	5-7
5.3.2 Evaluative design support tools.....	5-9
5.3.3 Information management systems.....	5-10
5.3.4 Other applications.....	5-12
5.3.5 Conclusions	5-12
5.4 Integrated Building Design System.....	5-13
5.4.1 Overview.....	5-13

5.4.2 Data exchange and building data modelling	5-17
5.4.3 Data exchange standards	5-21
5.4.4 Conclusions	5-24
5.5 Summary.....	5-24

CHAPTER 6

PROTOTYPE DEVELOPMENT, ANALYSIS AND SPECIFICATION

6.1 Introduction	6-1
6.2 Software Development Process	6-1
6.3 Concepts of COMBINE.....	6-3
6.3.1 Data integration and process integration	6-5
6.3.2 Building performance evaluation tools and design tools	6-6
6.3.3 Aspect model and integrated data model.....	6-8
6.4 Domain of the Prototype (MultiCAD)	6-10
6.4.1 Targeted users.....	6-10
6.4.2 Use scenarios	6-10
6.4.3 Design context	6-12
6.5 Design Request Analysis.....	6-14
6.6 System Functional Requirements.....	6-18
6.6.1 General requirements.....	6-18
6.6.2 Building energy consumption.....	6-18
6.6.3 Room daylight	6-19
6.6.4 Building element heat loss.....	6-19
6.6.5 Compliance checking	6-19
6.6.6 Overheating risk	6-20
6.7 Model Identification	6-20
6.7.1 BREDEM-8	6-21
6.7.2 Average daylight factor	6-21
6.7.3 Elemental U-value	6-22
6.7.4 Compliance checking	6-22
6.7.5 Summertime temperature	6-23
6.8 Data and Process Analysis.....	6-23
6.8.1 Primary I/O data	6-24
6.8.2 Unified data documentation	6-29
6.8.3 NIAM analysis.....	6-31
6.8.4 IDEF0 analysis	6-33
6.9 Summary.....	6-35

CHAPTER 7

PROTOTYPE DEVELOPMENT, DESIGN AND IMPLEMENTATION

7.1 Introduction	7-1
7.2 Criterion of Software Quality	7-1
7.3 Implementation Computing Environment.....	7-2
7.3.1 Hardware environment	7-2
7.3.2 Software environment.....	7-3
7.4 Object Oriented System Design	7-5
7.5 Data Model of MultiCAD	7-7
7.5.1 Features of the data model.....	7-16
7.6 System Structure of MultiCAD	7-19
7.7 User Interface Design.....	7-22
7.8 Data Mapping with the IDM	7-34

7.9 Coding Process	7-37
7.10 Discussions on the Implementation.....	7-47
7.11 Summary.....	7-48

CHAPTER 8

PROTOTYPE DEVELOPMENT, TESTING AND EVALUATION

8.1 Introduction	8-1
8.2 Overview of Software V&V	8-1
8.2.1 What is V&V?	8-1
8.2.2 Why is V&V needed?	8-2
8.2.3 When should V&V be applied?.....	8-2
8.3 Techniques of Software V&V	8-4
8.3.1 Walkthrough	8-4
8.3.2 Inspection.....	8-5
8.3.3 Testing	8-7
8.4 Testing Strategy for MultiCAD.....	8-8
8.4.1 Objectives and tasks	8-9
8.4.2 Test plan	8-9
8.4.3 Unit testing	8-9
8.4.4 Integration testing (stage 1)	8-18
8.4.5 Integration testing (stage 2)	8-21
8.5 Summary.....	8-27

CHAPTER 9

CONCLUSIONS AND ISSUES FOR FUTURE STUDY

9.1 Review of This Study	9-1
9.1.1 Overview of the technology transfer in the building industry.....	9-1
9.1.2 Identification of barriers of the technology transfer process.....	9-1
9.1.3 Development of an IBDS prototype	9-3
9.2 Issues for Future Study	9-5

APPENDIX A

I/O DATA DOCUMENTATION FOR DAYLIGHT CALCULATION

APPENDIX B

NIAM ANALYSIS

APPENDIX C

EXPRESS SCHEMA OF THE IDM

APPENDIX D

CLASS REFERENCE FOR MULTICAD DATA MODEL

REFERENCES

LIST OF FIGURES

Figure 1.1 Scheme and major issues of this study	1-5
Figure 2.1 Hierarchy of research activities	2-2
Figure 2.2 Hierarchy of knowledge.....	2-4
Figure 2.3 Relationship of the key concepts	2-5
Figure 2.4 General stages of research	2-8
Figure 2.5 Entries on energy in buildings in BRIX database.....	2-9
Figure 2.6 Relationship between research and practice	2-10
Figure 2.7 Knowledge cycle.....	2-10
Figure 2.8 Impact of different building stages on life-cycle costs.....	2-12
Figure 2.9 Participants of the building design.....	2-14
Figure 3.1 The balance between creativity and rationality	3-6
Figure 3.2 Decision making pattern	3-7
Figure 3.3 RIBA's plan of work	3-9
Figure 3.4 The Maven's map of the design process	3-10
Figure 3.5 Page's decision sequence model	3-11
Figure 3.6 Lockley's map of design process	3-12
Figure 3.7 Darke's map of the design process.....	3-14
Figure 3.8 The cyclic iterative process.....	3-15
Figure 3.9 Cyclic process model for problem solving	3-16
Figure 3.10 Average time devoted to design stages.....	3-20
Figure 3.11 Use of technical publications during building projects	3-21
Figure 3.12 The learning cycle.....	3-22
Figure 4.1 Components of a design support system.....	4-1
Figure 4.2 Two classification approaches	4-2
Figure 4.3 Classification of commonly used design support systems	4-3
Figure 4.4 Frequency of use of various technical publications.....	4-6
Figure 4.5 Interaction between designers and tools	4-12
Figure 4.6 Discrete provision of design support tools	4-13
Figure 4.7 Integrated provision of design support tools.....	4-17
Figure 5.1 Components of the computing environment.....	5-1
Figure 5.2 The structure of computer hardware	5-2
Figure 5.3 Change of computer cost and performance	5-3
Figure 5.4 Storage capacity of several types of disk.....	5-4
Figure 5.5 The penetration of computing into architecture firms	5-6
Figure 5.6 The proportion of articles on computing listed in the API	5-6
Figure 5.7 Multiple views of building produced by CAD	5-8
Figure 5.8 Example of computer evaluative program.....	5-9
Figure 5.9 Example of information management system	5-11
Figure 5.10 Structure of an integrated building design system.....	5-13
Figure 5.11 Two approaches to the development of IBDS	5-14
Figure 5.12 Database-Centred system.....	5-15
Figure 5.13 Executive-Centred system	5-16
Figure 5.14 Product Model centred system.....	5-17

Figure 5.15 Two data exchange approaches.....	5-18
Figure 5.16 Two levels of building data modelling	5-19
Figure 5.17 Illustration of the RATAS building data model.....	5-20
Figure 5.18 The RATAS model class hierarchy and relationships	5-20
Figure 5.19 The usage of RATAS model.....	5-21
Figure 5.20 The evolution of data exchange standard.....	5-22
Figure 5.21 Layer structure of STEP.....	5-23
Figure 5.22 Data exchange using STEP	5-23
Figure 6.1 Software development process and tasks.....	6-1
Figure 6.2 Structure of COMBINE project	6-4
Figure 6.3 Building life space and scheme of design support.....	6-5
Figure 6.4 The operation of a design tool	6-8
Figure 6.5 IDM and DTs in COMBINE.....	6-9
Figure 6.6 Architect at work.....	6-11
Figure 6.7 Structure of the I/O data description form	6-30
Figure 6.8 NIAM diagram of building model for daylight application	6-31
Figure 6.9 Different views of the building by different BPE tools.....	6-32
Figure 6.10 Basic structure of IDEF0	6-33
Figure 6.11 Average daylight factor.....	6-34
Figure 6.12 Determine window parameters	6-34
Figure 7.1 Data model for MultiCAD	7-10
Figure 7.2 An example of object nesting	7-17
Figure 7.3 Geometrical and topological modelling.....	7-18
Figure 7.4 System configuration of MultiCAD.....	7-20
Figure 7.5 Integration structure of MultiCAD	7-21
Figure 7.6 Data and methods in MultiCAD	7-22
Figure 7.7 User interface style	7-22
Figure 7.8 Interface structure of MultiCAD.....	7-23
Figure 7.9 Standard application window interface.....	7-25
Figure 7.10 Dialog interface style	7-26
Figure 7.11 Building Editor.....	7-26
Figure 7.12 Room editor.....	7-27
Figure 7.13 Wall editor	7-28
Figure 7.14 Site condition editor.....	7-29
Figure 7.15 Window editor.....	7-29
Figure 7.16 Frame editor	7-30
Figure 7.17 Glass editor	7-31
Figure 7.18 Daylight requirement dialog	7-31
Figure 7.19 Layer editor.....	7-32
Figure 7.20 Material Editor	7-33
Figure 7.21 Finish editor	7-33
Figure 7.22 Data mapping between MultiCAD and IDM.....	7-35
Figure 7.23 Visualisation of the IDM before the data mapping.....	7-36
Figure 7.24 Visualisation of the MultiCAD data model after the data mapping	7-37
Figure 7.25 Class hierarchy of the data model classes.....	7-38
Figure 7.26 Class hierarchy for the user interface classes	7-39
Figure 7.27 Room Editor user interface	7-40
Figure 7.28 The sequence of the daylight calculation.....	7-41
Figure 7.29 The interface for presenting the daylight calculation result	7-42
Figure 8.1 Software development and V&V activities	8-3

Figure 8.2 Sample checklist for requirement inspection.....	8-6
Figure 8.3 Two software testing approaches.....	8-8
Figure 8.4 MultiCAD testing procedure.....	8-9
Figure 8.5 The procedure of unit testing	8-11
Figure 8.6 Creating objects using MultiCAD interface	8-12
Figure 8.7 Inspecting object	8-13
Figure 8.8 Browsing the object methods.....	8-13
Figure 8.9 Executing object method.....	8-14
Figure 8.10 Debugger prompt is shown when an error is detected.....	8-15
Figure 8.11 Actor Debugger for program debugging.....	8-15
Figure 8.12 Interface for the elemental U-value	8-19
Figure 8.13 MultiCAD Layer editing interface.....	8-20
Figure 8.14 An example of stress testing errors.....	8-20
Figure 8.15 An example of additional safety checking.....	8-21
Figure 8.16 Multiple design tasks support 1	8-22
Figure 8.17 Multiple design tasks support 2	8-23
Figure 8.18 Multiple design tasks support 3	8-23
Figure 8.19 Data exchange demonstration 1	8-24
Figure 8.20 Data exchange demonstration 2.....	8-25
Figure 8.21 Data exchange demonstration 3	8-25
Figure 8.22 MultiCAD supplies general design information support.....	8-27
Figure 9.1 The completed MultiCAD prototype	9-5

LIST OF TABLES

Table 2.1	The evolution history of building practice	2-8
Table 2.2	Influencing factors in building design.....	2-13
Table 2.3	Technology transfer media and their characteristics.....	2-15
Table 3.1	Creativity in art, science and design	3-5
Table 3.2	Learning techniques of architects.....	3-23
Table 3.3	Usage of different information.....	3-25
Table 4.1	Computer usage in practice (% of all practices)	4-8
Table 4.2	Summary of current design support provision	4-9
Table 5.1	Generations of computer hardware	5-2
Table 6.1	Domain of the prototype	6-14
Table 6.2	Analysis of typical design requests in energy, compliance, daylight, overheating and U-value.....	6-16
Table 6.3	Data and process analysis task	6-23
Table 6.4	Input data comparison between different BPE methods.....	6-26
Table 6.5	Output data comparison between different BPE methods	6-28
Table 7.1	Example of conceptual mapping between MultiCAD and IDM.....	7-34

CHAPTER 1

INTRODUCTION

1.1 Introduction

This study is concerned with the problem of technology transfer from research to practice in the building industry, with a special focus on testing the feasibility of developing an integrated building design system. It is recognised that a large amount of scientific and technical information concerning the improvement of building performance has been accumulated in the research community, but has not been fully utilised in building practice. This under-utilisation of existing technology results in significant economic losses both in terms of research spending and building failures.

It is acknowledged that the rapid increase of computer application in the building industry provides a potential platform for future improvement in the technology transfer process. The study of computer based integrated building design systems has already become a focus of international research in recent years.

1.2 Background and Objective of the Study

Knowledge creation and accumulation are the result, but not necessarily the only objective of scientific research activities. At a seminar of the Science & Engineering Research Council, Cooper (1989) stated that the objective of research is "to advance knowledge and technological capability....[and]....to help achieve economic, social and cultural benefits." In this view, the potential benefits of any research achievement can only be realised through its application in practice. While it is debatable whether this applies to all research fields, it is nevertheless the case for applied research such as building science research.

Since the participants of research and practice are often separated in the building industry, there is a need for technology transfer from research to practice. It will be argued that, at present, there are many barriers in this technology transfer process. The research and practice communities have gradually realised the seriousness of this problem and a number of research initiatives have already been taken in related areas.

1.2.1 The context of this study

Before 1970s, the industrialised countries enjoyed relatively inexpensive energy supplies. Energy saving, in general, was not a significant consideration in building, manufacturing and other industries. The first energy crisis, in 1973, triggered a world-wide campaign of energy conservation. The building industry, as a major energy consuming sector, was rightly at the fore front of this campaign (Henderson, 1990). Energy conservation has

become a widely researched subject during the past two decades. National governments, through agencies, such as the International Energy Agency (IEA), have committed considerable resources to research into energy conservation in buildings. Significant achievements have been made in this area of research and some benefits have been demonstrated in practice. For example, according to a survey by Boyle (1989), the average total economic growth in all 21 IEA countries was 32% during 1973-1986. During the same period, the energy consumption in these countries increased by just 5%. The under proportional increase of energy consumption is mainly attributed to the use of energy conservation measures in these countries.

Despite some successes, the full potential of energy saving is yet to be realised (Newborough, 1991; Reddy, 1991). Today, the call for energy conservation has never been stronger. "Global warming" and "Greenhouse effect" are persistent headlines in the public media. Given the existing knowledge base of energy conservation techniques accumulated during the past few decades, the most pressing issue now is the transformation of these technologies into practical use.

Shove (1991) identified three phases in the international energy conservation campaign in the building sector:

- (1) In the early 1970s, the initial response to the energy crisis was to gather data and set targets for energy consumption.
- (2) The second stage was to develop and monitor methods and technologies which would help to reduce the energy consumption in the built environment. Much research effort was devoted to areas such as the use of solar energy, improved energy efficiency in buildings and so on.
- (3) The final stage is concerned with the dissemination of the research results or the process of technology transfer. Current practice in this respect indicates that considerable effort is still required.

There are no clear cut divisions among these three stages, there is just a gradual shift of emphasis. At the beginning, as a sense of urgency, various authorities set out standards on energy consumption. This was followed by numerous research projects, including theoretical analysis, modelling simulation and prototype testing. As a result of these efforts by the research community world-wide, a quantity of information has been produced and presented in all kinds of forms, i.e., textbooks, handbooks, reports, journal papers, as well as computer programs. A certain amount of this information has already got through to practice and considerable benefits have been yielded as a result. However, the major part of this output is still limited to the circulation within the research community.

For example, observations by Brown (1988) and Riddle (1988a) on the UK's passive solar design project, being carried out by the Energy Technology Support Unit have revealed the tasks and importance of the active promotion of technology transfer in the building industry. This project is aimed at the promotion of passive solar techniques as an aid to energy conservation in buildings. It includes a number of pilot design study projects during which designers and various experts work together to produce an 'optimum' design solution. During these pilot projects the problem of transferring passive solar technology into design practice has been highlighted. A large part of the available technical information is not used by designers due to reasons such as energy being a low priority in the design considerations, the lack of authoritative design guidance, inappropriate information presentation, etc. (Brown, 1988).

1.2.2 Current research in technology transfer

Since the early 1980s, technology transfer has been drawing increasing attention. In 1981 R.J. Appleyard at the opening ceremony of the first EEC Seminar on 'The Transfer and Exploitation of Scientific and Technical Information' pointed out that "one thing is certain we cannot afford to produce scientific and technical knowledge, an expensive raw material, without seeing that it is fully scrutinised for optimal use".

In the last few years, in addition to developing new scientific and technical information, research has also been focused on the dissemination of existing information to designers in practice. Many research projects have been carried out for this purpose, each of which has specific research objectives. These research initiatives can be classified into three broad categories:

- (1) Reviewing designers' information requirements. Designers have their own style of working and their information requirements are quite different from those of the researchers. The failure to acknowledge this difference contributes to the difficulties of information transfer to designers. Research in this category addresses issues such as "how do designers work?" "what kind of information do they need?" and "in what form do they need the information?". The research at the University of York in collaboration with the UK's Building Research Establishment (Marvin, 1985a) and the SOLINFO project of the European Community (Research digest, 1988) are examples of this type.
- (2) Producing design advice. The aim of research projects in this category is to produce correct, concise, relevant and authoritative design guidelines and advice based on the existing technical information. Examples of this type of research are the Energy Technology Support Unit's *Passive Solar House Design* study (AA, 1988; BRE,

1988) and the *Energy Efficiency Best Practice Programme* of the Department of Energy (EEO, 1989).

- (3) Developing design support tools by exploiting developments in modern Information Technology (IT). IT has the potential to be superior to traditional information transfer media, such as printing materials, in a number of crucial respects such as information storage capacity, processing speed, user interaction and so on. Research in this area is to explore ways of exploiting these advantages to promote technology transfer from research to practice. The Europe project, **Computer Modelling in Building Industry in Europe (COMBINE)** (Augenbroe, 1989a), is one example in this category.

1.3 Conduct of this Study

The main objective of this study is to investigate the problem and possible improvement of technology transfer process from building research to architectural practice. The hypothesis is that a computer based Integrated Building Design System (IBDS) can be developed which will improve this process by providing design support to designers at the point of use and meeting design requirements. For this purpose, the IBDS will need to support multiple design tasks and operate interactively under the designer's control.

In order to achieve the above objective, this study set out to:

- Review technology transfer situation in the building industry, especially the design sector, and to address the requirements and barriers to this process.
- Study the information requirements of architects in practice on the basis of existing studies in this area.
- Analyse existing design support measures, paying special attention to the deficiencies of these measures from the architects' view point.
- Define the requirements of an integrated building design system.
- Explore the potential of an IBDS for the improvement of design support and the technology transfer process.

The scheme and the major issues of this study are illustrated in figure 1.1. The study can be divided into two parts. The first part is to review and analyse the current situation of technology transfer from research to practice in the building industry. Issues in the first part include an analysis of the gap and barriers between design practice and research; designers' needs in terms of technical support as well as the review of existing support measures.

The second part of this study is devoted to the development of a computer based prototype of an integrated building design system. During the prototyping process, issues of common concern, which have no clear answers yet, will be explored. For instance, "how to map designers' requirements to the design support system?" "how to make the integration of multiple tools work?" "how to develop a neutral description of the building?" The prototyping phase is considered as an essential step to understand how the conceptual requirements for design support can be translated into existing computer technology and the constraints that will be generated. Given the magnitude of this task, the prototype described in this thesis can only be viewed as a contribution towards the long term goal of the future development of computer application in architecture.

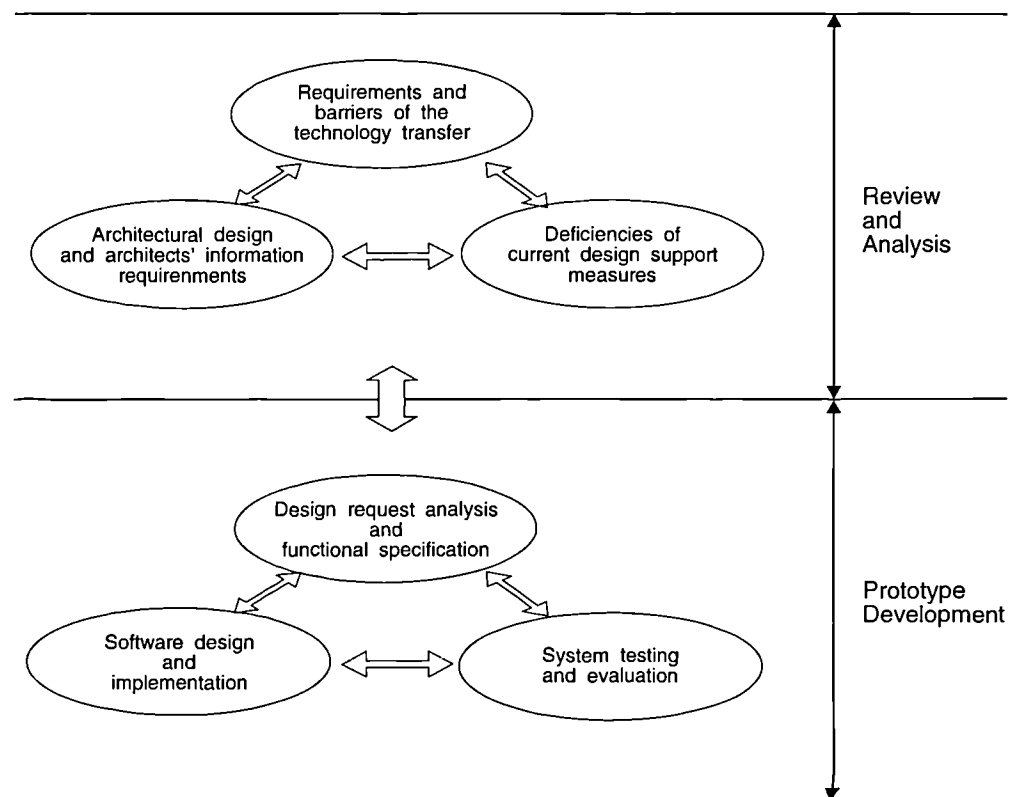


Figure 1.1 Scheme and major issues of this study

1.3.1 Review and analysis

The potential benefit of scientific and technological achievements can only be realised through their widespread use. In the case of energy conservation in buildings, the related expertise developed by researchers is a transferable commodity. However, if it cannot be accessed and readily used through the architects in practice, it is effectively wasted as a product staying permanently in the warehouse. Unfortunately, at present due to various

reasons, these potentially beneficial technologies have not found their full use in the building practice (Brewer, 1988b).

One of the major causes for the under-utilisation of existing technologies is the discipline gap between researchers and practitioners, architects in particular. Acknowledging this problem, the questions that researchers have to address and answer are: firstly, can the research community carry out research that meets the needs of practice? secondly, can the research community present the research findings in an appropriate form to practitioners?

Correct understanding of the nature of architectural design and its practice is essential to successfully answering the above questions. Architectural design is a complex process and it involves decision making and decision evaluating. Therefore, the information designers need during the design serves two purposes: to help them generate design ideas and to help them evaluate the ideas. In this sense, design support can be classified into two types:

- Generative support. This includes materials, such as design advice, guides, handbooks, etc., which are directed towards helping designers make design decisions.
- Evaluative support. These are tools, i.e., algorithms, tables, monograms and so on, which help designers evaluating their design decision against specific criteria.

Architectural design is often an interactive process. Unlike, experts of specialist domains, such as structural and lighting engineers who can concentrate their attention on one aspect of building design, architects have to take all considerations into account interactively in the context of the whole design. They have to balance various factors dynamically during the decision making process (Bryan, 1986). Therefore, technical information support for a single design aspect is far from the ideal situation.

At present, design support information is commonly distributed as printed materials. This type of technical support requires considerable effort and competence on the part of the information receivers in order to make the knowledge transfer successful. Unfortunately, many architects are not good at absorbing technical information, or they do not have enough time to do so (Marvin, 1985a). The current scientific rather than practical oriented presentation format of the majority of the technical publication does not help the architects to assimilate and use the information content. In addition, printed materials are not very suitable to present evaluative design support tools, i.e., calculations, due to the lack of interaction with the user.

1.3.2 Prototype development

With the rapid development of computer hardware and software, computer applications have received increasing attention. They offer many advantages over traditional printed

materials in key areas, such as information storage, information retrieval and the handling of evaluative calculations.

The second part of this study is the development of a prototype design support system for the early building design stage undertaken as a part of the EEC funded COMBINE project (Augenbroe, 1989b). It will be argued that the design support system will need to:

- support multiple design tasks,
- be responsive to designers' requests,
- exchange information between different tasks, and
- be extendible to include more design support tools.

Given the focus of the COMBINE project, the domain of the prototype is energy related issues at the early stage of building design. The anticipated design context is an architect working on the specification of the external building elements. The prototype will cover some important aspects of building design criteria, i.e., building energy consumption, elemental heat loss, room daylight, overheating and building compliance checking.

In order to ensure that the prototype is responsive to the designers' needs, a design request analysis has been conducted, which identified several generic types of design request. On the basis of the design request analysis, the functional requirement of the prototype is set out and several building performance evaluation methods are selected.

The design and implementation of the prototype set out to meet the functional requirements. An Object Oriented (OO) paradigm is adopted during the development process and the advantages of the OO approach over the traditional procedural based approach in software engineering are argued due course. The primary objective of the prototype development is to explore and to test the concept of an integrated building design system. Developing commercial software for architectural design offices is beyond the scope of this study.

1.4 The Thesis

The thesis is organised in nine chapters. This introductory chapter describes the general background of this study and the remaining chapters address the following issues.

Chapter 2 examines the needs for technology transfer from the research community to architectural practice. Several barriers in the transfer process are identified.

Chapter 3 analyses architectural design. Issues discussed in this chapter include the nature of design, the design process, how do designers work, what support do they need and in what form should it be provided.

Chapter 4 reviews existing design support measures and analyses their deficiencies against designers' needs.

Chapter 5 introduces a general account of the state of the art in the development of computer application in architecture, integrated design support systems in particular.

Chapter 6 begins the descriptions of the prototype development, namely MultiCAD. This chapter deals with the domain of the prototype, the functional requirements, the calculation methods as well as initial data and process analysis.

Chapter 7 presents the design and implementation process of MultiCAD, including the establishment of a building data model, system structural design, interface design and coding.

Chapter 8 is devoted to system testing and evaluation of MultiCAD.

Chapter 9 summaries this study and explores some issues for future research in this area.

CHAPTER 2

TECHNOLOGY TRANSFER FROM RESEARCH TO PRACTICE

2.1 Introduction

Academic research and practice in the building industry have become two largely separated activities with few overlaps between them. In general, research is carried out by academic bodies such as building science research institutes and universities, while practice is carried out by architects, planners, developers, builders and so on. The process of putting research results into practical use is known as 'technology transfer', 'information transfer' or 'knowledge transfer'. For many reasons this transfer process in the building industry is not a smooth one at present. This chapter reviews this area in order to assess the problems and difficulties involved. Following a definition of terms, the issues to be addressed include:

- a brief overview of research and practice in the building industry,
- the need and key actors of the technology transfer process,
- technology transfer media and influencing factors,
- existing barriers to the technology transfer process, and
- suggestions on future improvement measures.

2.2 Definitions

"Technology transfer", "Information transfer" and "knowledge transfer" are often used by different authors to describe similar and sometimes different processes of putting research achievements into practice. There is the potential for confusion due to the different meanings assigned to the terms "research", "practice", "technology", "knowledge" and "information". Definitions of these terms are given to ensure a consistent view in the context of this study.

Research: Research is an activity designed for the disclosure of certain unknown aspects of the universe and the acquisition of new knowledge. Each research study undertaken is into some unknown aspects of the universe. The researchers engaging in this activity usually explore the unknown using systematic methods and strategies.

There are many kinds of research activity, e.g., research in pure physics, research in applied physics, research in building science, research in product manufacture, etc. Apart from the differences in subject matters, these research activities also explore the universe at different levels of abstraction. For instance, research in pure physics studies the fundamental principles of the universe, and many research subjects in this area are beyond ordinary people's comprehension. On the other hand, research in product manufacturing is closer to people's daily life and addresses the world at a more general level without

involving its subtle underlying principles. From this view point, research activities can be divided into a three-level hierarchy (figure 2.1).

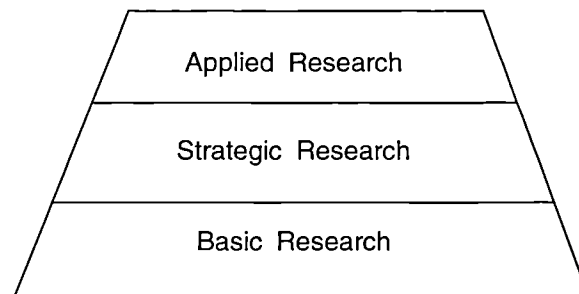


Figure 2.1 Hierarchy of research activities

The Universities Funding Council, in its research assessment exercise circular letter, has cited the following definitions of basic, strategic and applied research from the *Frascati Manual and Cabinet Office Annual Review of Government Funded Research and Development*:

- **Basic research** is experimental or theoretical work undertaken primarily to acquire new knowledge of the underlying foundation of phenomena and observable facts, without any particular application or use in view.
- **Strategic research** is applied research which is in a subject area which has not yet advanced to the stage where eventual applications can be clearly specified.
- **Applied research** is work undertaken in order to acquire new knowledge. It is, however, directed primarily towards practical aims or objectives." (Universities Funding Council, 1992)

It has to be acknowledged that the division between research types is not clear cut and figure 2.1 is only a symbolic representation. Building science research is essentially in the domain of the applied research. This is based on the observation that the objective of most of the research activities in this area is to promote direct benefits to society by improving the built environment, reducing the building capital cost, reducing energy consumption and so on.

Practice: Research provides understanding of the universe and potential ways to change the universe for the ultimate benefit of the mankind. Practice is the activity to make changes upon either physical or abstract aspects of the universe. For example, the practice of architects and builders is the physical creation of buildings. The practice of financial advisers is the provision of advice to their clients. The relationship between research and practice could be described as research providing guidance for practice; practice utilising

and exploiting the research findings as well as defining the context for new research. However, there is no one-to-one mapping relationship between research and practice. Some research results can be used by several practice fields and one practice might need inputs from several research fields. Generally, research at the applied level in the research activity hierarchy is more easily used in practice, while by definition, basic research and possibly strategic research are unlikely to find direct use in practice.

Practice in the building industry is a very complex process. The aim of this process is the creation of buildings on real world sites in order to satisfy the needs of human beings. This process involves not only the construction work on the site but also activities directly linked to the construction, such as planning, design and project management. Therefore, many parties are involved apart from the builders, for example, government, planners, designers, occupants and sometimes building researchers. Building practice has many requirements and influential factors. Very often, people who are directly involved in the process can not rely on their direct personal knowledge and experiences alone. The fact that they need help from outside such as the research community underlines the need for communication between research and practice.

Knowledge: As discussed above, research is the activity to explore unknown aspects of the universe. Knowledge is one's understanding of known aspects of the universe. The meaning of the term knowledge is something people often take for granted yet there is no universal definition for it. The Oxford English Dictionary (1967 edition) defines knowledge as "the fact of knowing a thing, state, etc., or person." Definitions of knowledge are also suggested by others:

"Knowledge is the symbolic representation of aspects of some named universe discourse" (Frost, 1986).

"Knowledge is the amount of facts ('what') and their interrelationships ('how and why') that exist within the human brain" (Johnson, 1987b).

Many other definitions can be found of a similar nature but it is perhaps more helpful to analyse the meaning of "knowledge" than to seek further definitions.

- (1) Knowledge is an accepted explanation of certain aspect of the universe at a certain time. As human exploration of the universe proceeds, existing knowledge might be proven wrong. For instance, people first believed the earth to be the centre of the universe, then the sun to be the centre. Now both of these once accepted items of knowledge have been proven to be incorrect.
- (2) Knowledge is a human being's understanding of the universe. It is something that exists in an individual's brain. To one individual, he or she does not have the knowledge unless it exists in his or her brain. The main purpose of education in modern society is to equip people with existing knowledge.

- (3) The universe can be disclosed at different levels. Therefore knowledge can also be classified into different levels. The classification of knowledge can be done in many ways depending on the purpose. For example, Hofstadter (1985) defined declarative knowledge and procedural knowledge for computer systems. Karlen (1987) divided knowledge into pre-paradigmatic scientific knowledge, paradigmatic scientific knowledge, tacit (implicit) knowledge and ordinary explicit knowledge.

Corresponding to the classification of the research activities (figure 2.1), knowledge can also be classified into three levels, scientific knowledge, technical knowledge and practical knowledge as shown in figure 2.2.

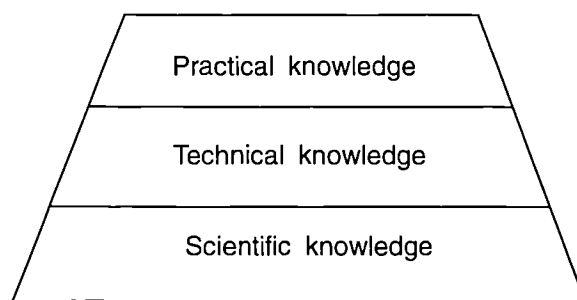


Figure 2.2 Hierarchy of knowledge

It is important to notice the difference and relationship amongst these different types of knowledge. Scientific knowledge is gained by scientific methods and expressed in scientific language, while other kinds of knowledge are usually developed on the basis of scientific knowledge. For example, the scientific foundation for the calculation of heat loss in building, is Fourier's Law. Some simple calculation models are derived from it, i.e., ESP and BREDEM. Further simplification is possible, for instance, the statement "in order to keep the building energy consumption within an acceptable limit the U-value for its elements should not be greater than such and such".

People in building practice often have difficulty in understanding material which is written in scientific language, i.e., the description of Fourier's Law or even the background description of ESP and BREDEM. However, they can accept practical knowledge much more easily, such as the U-value limits for building elements. In this example, the principle of Fourier's law is scientific knowledge; calculation methods such as ESP and BREDEM are technical knowledge; while simple U-value requirement can be considered as practical knowledge. Some of the existing difficulties in technology transfer are caused by disregarding the distinctions between scientific knowledge, technical knowledge and practical knowledge (section 2.8.3).

Technology: technology is a subset of knowledge which deals with practical application. It is at the top end of the knowledge hierarchy and it is more closely related to the real world of practice. Technology is often developed on the base of other types of knowledge. For example, energy conservation technologies in buildings are based on thermal theories, knowledge of material performance and so on.

Information: Information has a similar meaning to knowledge. The difference between knowledge and information is that the former describes the state of existence while the later emphasises the communication of fact. For example, Fourier's Law, the models for ESP and BREDEM are knowledge. When they presented in paper or computer programs, in the receivers' view they become information. Johnson (1987b) defined information as knowledge "available in a communicative form (i.e. coded into symbols that may be used for communication)".

There are two layers of meaning in this definition:

- (1) Information contains a piece of knowledge.
- (2) It also implies the communication media of this knowledge, i.e., speech, written language, pictures, computer files etc.

Relationship among these key concepts: The relationship among these key concepts can be summarised as follows, knowledge is the basic terminology used to describe the achievement of human research activities. Knowledge presented in a communicative form is considered as information; while technology is a subset of knowledge which deals with practical applications.

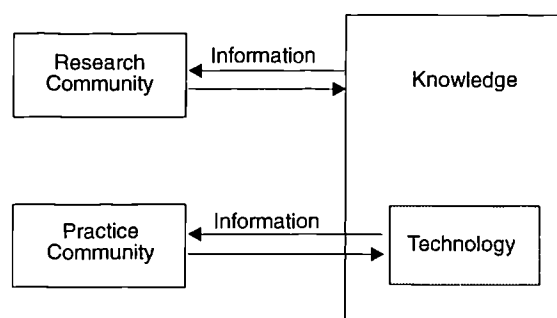


Figure 2.3 Relationship of the key concepts

Figure 2.3 illustrates this relationship. Communication in the research community can be very different from that within practice or between research and practice. In general, the research community as a whole can understand and is able to communicate on a wide and deep knowledge base. People in practice are usually only able to understand a subset of

the knowledge base or the practical technology in their field. Research results presented in a scientific form in professional journals are very unlikely to be used directly by architectural practitioners. To acknowledge this difference is very important if the communication problem between researchers and practitioners is to be solved.

In the following discussion, 'knowledge transfer' and 'technology transfer' are used with the same meaning that is to transfer appropriate knowledge from research to practice.

2.3 Building Research and Practice

This section introduces a general review of research and practice in the building industry as well as their inter-relationship. Building practice can be dated back to pre-history, while identifiable building research is relatively new and was only separated from the practice as an independent discipline early in this century (Groak, 1992).

2.3.1 The separation of research and practice

Building shelter is one of the earliest skills learnt by our ancestors. However, until the beginning of this century, building design and practice primarily concentrated on aspects of building form such as external appearance and internal spatial relationships. The understanding of building behaviour on physical aspects, such as, thermal, light, acoustic, was very limited. Things have changed during the course of this century. Due to the advance in science and technology, especially, the application of new materials in building construction, people have gradually realised the possibility of control over the performance of the built environment. An increasing need has emerged for the study and research into all aspects of the building environment (Groak, 1992).

During the early part of this century, the pioneers of the modern architecture movement broke away from traditional building practice. They did not believe that a building's functional, economical and other considerations should be compromised for the sake of building form. On the contrary, the new fashion of building practice emphasised the functional side of the building environment. In the declaration of this movement, Le Corbusier (1976) stated in his famous slogan that "a house is a machine for living in". According to him, the standard for judging buildings should no longer be mass, rhythm, proportion and style, but their ability to provide people with sunshine, warmth, pure air and a clean floor. In addition, buildings must be economic and suitable for mass production. There was a complex social background behind the modern architecture movement. The most influential factors were the massive building demands generated as a result of the industrial revolution and the advance of modern technology especially the application of

new building materials such steel and concrete. Although, later, modernism has been accused of fostering inhuman qualities, it nevertheless fundamentally changed building practice.

Increasing demand has made building practice an ever complex process. Structure, light, thermal, cost and so on, are increasingly being studied by specialised professionals. Architects alone are no longer able to handle all the problems concerning building design and construction. Against this background building research has separated from building construction practice as an independent profession. In the last thirty years, large numbers of building research institutes have been set up in many countries, and building science sections have been established in many schools of architecture in universities. This separation and expansion of the research activity have brought about an explosion of new knowledge in building science and related areas.

"Unfortunately, this is only one side of the picture. As has happened in other areas, the growth of research as a specialised and specific activity has created circumstances which makes the effective use of its results more difficult. Research tends to become isolated and satisfying in itself. Knowledge, and the status of research workers, advance, and those their work was originally meant to help - designers, builders, users - are left to make what they can out of the results" (ART, 1970).

In the last two decades, the issue of transferring research results to practice began to receive attention. However, today, the problem still remains (Hall, 1989).

2.3.2 Building research

Building research is an applied research field which deals with knowledge creation in the building industry.

The motivations behind the work conduct of individual researchers are complex. Reizensteins (1975) suggested that "people do research for two reasons: first because it is interesting, and second because it may be useful". He acknowledged that this issue is sometimes complicated by the fact that useful knowledge is not always interesting, and interesting knowledge is not necessarily practically useful. While interest might be a desired attribute of a research subject, an essential attribute of research, particularly applied research, is its usefulness. Most research activities carried out by corporate bodies and institutions usually involve significant financial expense and the anticipated benefits often become essential preconditions to the funding of many research projects.

While every research project may have its own specific approach, most of the research activities tend to follow a similar pattern. The process starts with the formulation of the problem which is followed by hypothesis, experimental design, experiment or study, analysis, synthesis and conclusion. This process is illustrated by figure 2.4.

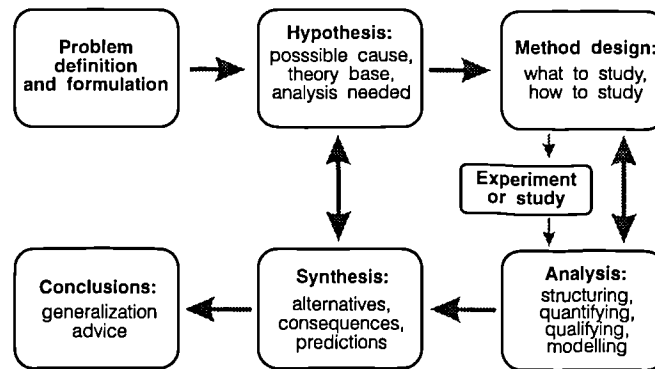





Figure 2.4 General stages of research (after Johnson, 1987b)

The research activity is rarely a linear process. There are various returning loops to previous stages and figure 2.4 only shows a limited local view of the research activity. It does not show relationships with outside world from which the research problems are derived and where the conclusions are feedback in the case of applied research.

2.3.3 Building practice

The long evolution history of building practice can be broadly divided into three periods, each of which has distinguishing characteristics as shown in table 2.1.

Table 2.1 The evolution history of building practice

Period	Participants	Characteristics	Symbolic Product
Stone age	individuals	Building practice is simple, the user is the designer and builder.	
Pre-modern time (pre 1900)	master builders, building workers, users	Design becomes an independent profession. Designers are closely involved in the building process. Design consideration focuses on the aesthetic aspect of building forms.	
Modern time (post 1900)	planners, architects, technicians, specialists, consultants, developers, subcontractors, users and so on	Building practice is a complex process. It consists planning, design, construction and maintenance. The designer is the leader of the design team, design involves technical aspects as well as aesthetics.	

The main driving force behind this evolution is the development of social, economical conditions, particularly science and technology, as well as the corresponding human demands. Nowadays, building practice has become more and more complex requiring a greater range of skills and an increased number of participants. For example a modern computing laboratory has quite different and, in a sense, more sophisticated demands than a medieval palace. Compared with ancient times, many factors have been taken as important considerations in modern building practice, i.e., economic, short construction duration, occupant's satisfaction, etc.

2.4 The Need for Technology Transfer

There has been a rapid increase in knowledge on many aspects of the building environment in the last few decades. People usually describe this situation as a "knowledge explosion". For instance, a literature search using the BRIX on-line computer data base shows that there has been a big increase of publications dealing with energy in buildings since early 1970s (Wiltshire, 1988). It has been found that there have been five hundred or so articles appeared on peer-refereed English language journals alone every year (figure 2.5).

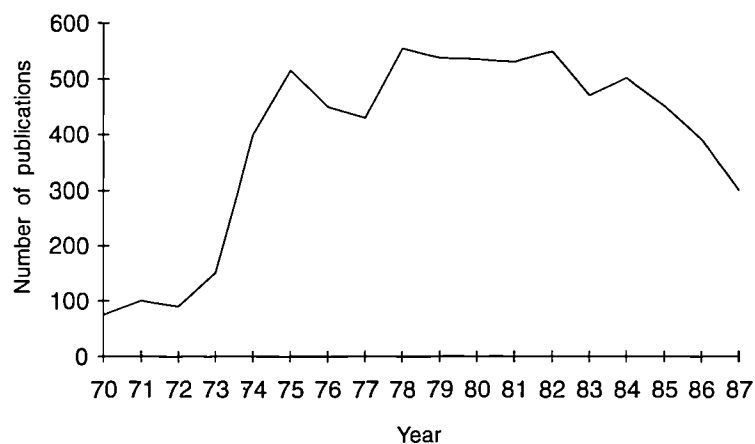


Figure 2.5 Entries on energy in buildings in BRIX database (after Wiltshire, 1988)

Figure 2.5 suggests a slight decline in the number of publications concerning energy in buildings in recent years. The exact reason for this is not clear, but Wiltshire (1988) offered two possible explanations. First, the pressure for energy conservation is not as great as it was in the early 70s. Second, the knowledge in this area has matured, a reason also suggested by Shove (1991).

The discussion in section 2.2 outlined the basic relationship between research and practice, that is, research creates new knowledge while practice makes use of the existing knowledge. Johnson (1987a) illustrated this relationship in figure 2.6.

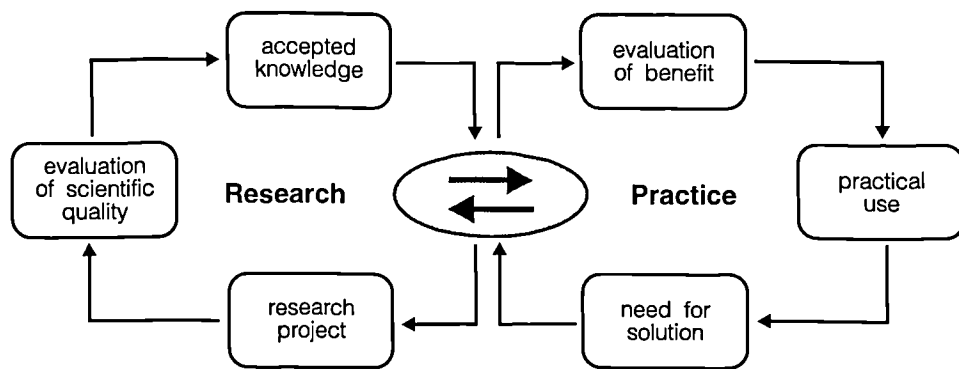


Figure 2.6 Relationship between research and practice (based on Johnson, 1987a)

The grey area in Figure 2.6 represents the communication link between the research and practice communities. It includes a two-way information flow. Research projects are defined based on the needs of practice. The knowledge produced by research provides the solutions to the practical problems.

This basic principle maps with the knowledge cycle proposed by Karlen (1987). According to him, the knowledge cycle composes of three processes, knowledge creation, knowledge dissemination and knowledge utilisation (figure 2.7). It is a cyclic and continuous process. From knowledge creation to knowledge utilisation is only one cycle in the process. During the knowledge utilisation phase, new problems will occur and new research initiatives will be proposed. Subsequently, the knowledge cycle will be repeated at a higher and more informed level.

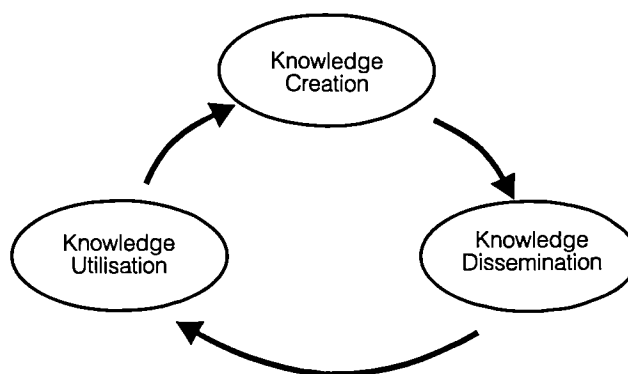


Figure 2.7 Knowledge cycle (after Karlen, 1987)

There are two important aspects in the knowledge cycle or the technology transfer process. One is the time lag from knowledge creation through to its utilisation, and the other is the extent of the knowledge dissemination and utilisation. The essence of technology transfer requires the dissemination of knowledge to the widest possible coverage within the identified target audiences with the shortest time delay.

Unfortunately, there are evidences which indicate that the knowledge transfer process in the building industry does not operate satisfactorily (Brewer, 1988b; Cooper, 1984; Diepeveen, 1987). Brambley (1990) argued that energy consumption in building environments could be considerably reduced by applying existing knowledge. Many other building defects, such as poor sound insulation, draught, cold and so on, could also be avoided. These potential benefits further highlight the need for improvement in the technology transfer process.

The evident under-utilisation of existing knowledge in the building industry requires an examination into the technology transfer process. The investigation must address the following issues.

- Who are the most important actors or participants?
- What are the means currently used for technology transfer?
- What are the influencing factors for success or failure?
- What are the existing barriers to the technology transfer process?
- What can the participants involved do to improve the situation?

2.5 Key Actors in the Technology Transfer Process

There are many descriptions of the building life cycle, and in the UK, the RIBA's plan of work is one of the best known (RIBA, 1988). According to this plan, a building experiences several phases in its life cycle, from brief, design, construction, maintenance through to demolition. Dupagne (1991) suggested that along the time scale, the design and construction process can both take 1-5 years. Building use and maintenance could last 100 years or more. The total cost during maintenance could be 2-3 times higher than costs of design and construction put together. However, this does not imply that maintenance is the most important stage for reducing building life-cycle costs and to improve building performance. Dell'Isola (1982) presented the impact on building cost of each building phase, in which the design stage has been identified as having the most impact apart from the owners' requirements (figure 2.8).

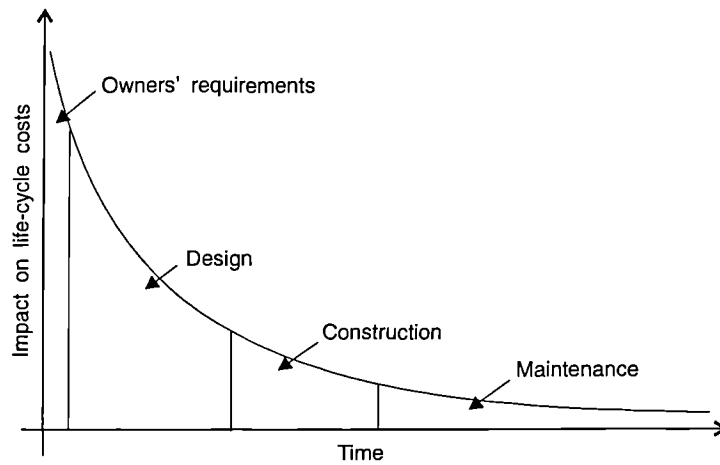


Figure 2.8 Impact of different building stages on life-cycle costs (after Dell'Isola, 1982)

Further investigation of building design itself reveals its complexity with numerous influencing factors, ranging from social, economical, physical to administrative ones, as well illustrated by Brambley (1988) in table 2.2.

Table 2.2 Influencing factors in building design (after Brambley, 1988)

Human Factors	Physical Factors	External Factors
<ul style="list-style-type: none"> Activities Behaviour Objectives/Goals Organisation <ul style="list-style-type: none"> -Hierarchy -Groups -Positions -Classifications -Leadership Characteristics (Demographics) Social Forces Political Forces Interactions <ul style="list-style-type: none"> -Communication -Relationships - Transfer of materials, etc. Policies/Codes Attitudes/Values Customs/Beliefs Perceptions Preferences Qualities <ul style="list-style-type: none"> -Comfort -Productivity -Efficiency -Security -Safety -Access -Privacy -Territory -Control - Convenience 	<ul style="list-style-type: none"> Location <ul style="list-style-type: none"> -Region -Locality -Community - Vicinity Site Conditions Building/Facility Envelope Structure Systems <ul style="list-style-type: none"> -Engineering -Communications -Lighting - Security Space <ul style="list-style-type: none"> -Types -Dimensions - Relationships Equipment/Furnishing Materials/Finishes Support Services <ul style="list-style-type: none"> -Storage -Parking -Access -Waste -removal - Utilities(water, sewage, telephone) Uses Functions Behaviour/Activity Settings Operations Circulation Environment <ul style="list-style-type: none"> -Comfort -Visual - Acoustical Energy Use/Conservation Durability/Flexibility 	<ul style="list-style-type: none"> Legal Restrictions(Codes/Standards/Regulations) <ul style="list-style-type: none"> -Building -Land use -Systems -Energy -Environment -Materials -Safety - Solar access Topography Climate Ecology Resource Availability Energy Supplies/Prices <ul style="list-style-type: none"> - Conventional - Solar - Alternatives Economy Financing Time <ul style="list-style-type: none"> -Schedule -Deadlines - Operations Costs/Budget <ul style="list-style-type: none"> -Construction -Materials -Services - Operations Costs/Benefits

At present, the increasing complexity of the design problem makes it impossible to be handled solely by the architectural profession. An increased number of experts are coming into the design process, and relatively, the architects' role has been declining from 100% in 1919 to an estimated 38% in 1979 (Figure 2.9). Nevertheless, they are still by far the most important actor in building design. The combination of the importance of design phase in the building life-cycle and the importance of architects in the design process makes them one of the most, if not the most, important target for the technology transfer in the building industry.

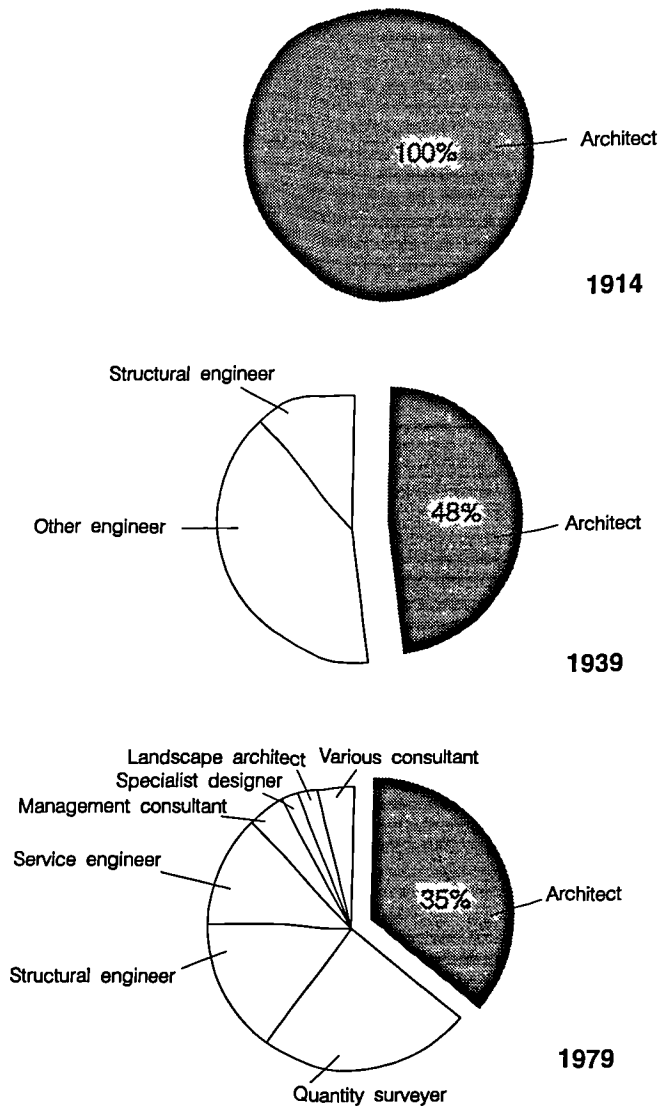


Figure 2.9 Participants of the building design (after Reynolds, 1980)

2.6 Technology Transfer Media

There are many technology transfer media available, such as books, research reports, journal papers, visual materials, computer programs, personal communication and so on. There is no single rule to judge which is the most appropriate media for the communication of knowledge and the choice often depends on individual circumstances. For example, for widely demanded knowledge such as general energy conservation advice, the best choice might be handbooks, for calculation methods computer programs are more suitable, and for specific issues, i.e., solutions to individual problems, the most suitable means of communication might be by personal contact.

Nevertheless, several criteria can be identified for judging the effectiveness of technology transfer media.

- (1) **Time delay:** It means the time needed to get the knowledge to its intended target audiences. Obviously, the shorter the time delay, the better it is.
- (2) **Accessibility:** This is about the number of people who can access a particular communication media. Technology transfer in building industry requires maximum access to the up-to-date technology by the identified target audience.
- (3) **Cost:** Cost implies the expense involved in transferring the technology to its users.
- (4) **Assimilation.** The objective of technology transfer process is not simply to give the users physical access to the knowledge, but more importantly to enable them to understand and make use of it. Assimilation is a measure of the effectiveness of comprehension and usability.

Although no explicit data is available, it is possible to make a subjective analysis of the commonly used communication media against the above criterion. Table 2.3 shows the author's personal quantitative evaluation. In the table the number of '*' indicates the performance of the communication media; the more '*' a media has, the better it is in that particular aspect, vice versa. For instance, books are very poor in respect of time delay, and relatively poor in terms of assimilation, but very good in terms of accessibility and cost.

Table 2.3 Technology transfer media and their characteristics

	Time delay	Accessibility	Cost	Assimilation
Books	*	*****	*****	**
Conference proceedings	**	*****	*****	**
Journal articles	**	****	****	*
Computer applications	**	***	***	***
Slides, microfilms and video	**	***	**	**
Conference presentations	***	**	**	***
Exchange of drafts	***	*	**	***
Seminars	***	**	**	****
Working messages	****	*	*	*****
Personal talks	*****	*	*	*****

It has to be stated that table 2.3 is not an exclusive list, nor is it based on solid statistics. It is used to show the characteristics of various technology transfer media. The performance ranking in the table is a relative one, it does not indicate any quantitative measurement. For example, it takes less time to transfer knowledge through a personal talk than through books; this is illustrated in the table. However, the table does not imply that personal talk is five times better than books in terms of time delay. Similarly, books are superior to

personal talks in the aspects of accessibility and costs, but are poorer in term of assimilation.

The communication means shown in table 2.3 can be roughly divided into two types, those that require direct personal contacts and those that do not. "Personal talk", "Work message", "Seminar", "Exchange of drafts" and "Conference presentations" belong to the first type. They are effective and can transfer the most up-to-date research findings. However, because they require face to face communication, the audience who can benefit is usually limited. They are more expensive in transferring technology to mass audiences.

In practice, technology transfer to mass audiences usually relies on means which do not require direct personal contact, such as "Books", "Conference proceedings", "Journals", "computer applications" and "Slides, microfilms and video". The advantage of these communication means is that they are less expensive and can be distributed widely. However, their biggest disadvantage is that it is very difficult for these media to improve assimilation due to the lack of interaction with the receivers. In addition, the time delay is usually longer since they usually go through the process of editing, printing and publishing.

Among these media, computer applications have increasing significance and great potential for improvement in almost every aspect. Although the initial development of computer software may take just as long as publishing books, its subsequent updating usually requires a shorter time scale than revising a book. There is evidence showing that computers are increasingly adopted by architectural practices (Ray-Jones, 1990). This trend will inevitably lead to a wider access by architects to computers and computer applications. In the mean time, the capital costs and operational costs of computer application are coming down and the assimilation of knowledge is being improved by 'user friendly' software applications. Computer application in architecture will be the theme of discussion in chapter 5.

2.7 Influencing Factors

There are many factors that have impact on the success or failure of the technology transfer process. The prominent factors include personal attitudes and skills as well as the communication technology which is adopted.

2.7.1 Attitudes and skills

Knowledge transfer from research to building practice requires extensive communication between researchers and designers. There is a natural discipline gap between these two

groups of people. They have different skills, different working approaches, different learning styles and different professional languages (Powell, 1988). This makes it even more important for both researchers and architects to show positive attitudes towards the importance of the knowledge transfer process. Unfortunately, in reality, this is not always the case. In many cases neither researchers nor architects claim the responsibility and are willing to commit themselves to the knowledge transfer.

On one hand, researchers are keen on achieving higher professional standards for their work, paying little attention of the transfer of the existing research findings to the practice world (Richardson, 1990). On the other hand, architects tend to rely on their previous experiences rather than to look for new knowledge when they confront new problems.

In addition, the success of knowledge transfer does not solely depend on the willingness of the knowledge senders and receivers, certain skills are also required from both sides of the transfer process. In the case of building research and practice, researchers should know what architects need, how they need it and how their needs can be met. On the other hand, architects should have the knowledge of what new technology is available concerning their problem and how to get access the technology.

2.7.2 Communication technology

Another influencing factor of technology transfer is the communication technology or communication media employed (section 2.6). It could have a decisive impact on the time delay and the scope of the knowledge dissemination. In this respect, the development of modern Information Technology (IT), especially the increasing use of computers in building practice, provides a promising prospect for the future (chapter 5).

2.8 Existing Barriers

At present, there are barriers on the technology transfer process at each stage of the knowledge cycle, knowledge creation, dissemination, utilisation (figure 2.7), as well as at the research administration.

2.8.1 Problems with administration

Problems in this aspect are mainly reflected by the lack of an overall strategy for knowledge transfer from the organisations who initiate and fund building research activities. Many research projects in the building industry are sponsored by government organisations. The common practice is that research institutes carry out the study, then the research findings are reported to the sponsors and sometimes published in academic

oriented journals. It is very rare that the requirement of broad dissemination of the research results to practice is included as part of the research contract (SERC seminar, 1989).

2.8.2 Problems with knowledge creation

As discussed above, building researchers and architects are two influential groups in building research and practice. The current educational system determines that most of the architects do not engage in research concerning building performances. The majority of building researchers come from disciplines other than architecture, such as physics, civil engineering and so on. It is not the objective of this study to judge whether this situation is good or bad. The existence of this discipline gap between researchers and architects does however contribute to the communication difficulties between these two key professions.

One result of the lack of effective communication between researchers and architects is the imbalance of research topics. Brewer (1986) pointed out that current research into building design was physical/techno-economic oriented. The hidden social factors, the needs of designers and building occupants, were rarely studied by researchers. As he put it:

"i. The needs and habits of our target audience(s) have rarely been considered, either in the conduct of research or in the preparation of its findings for dissemination. So too little thought has been given to the people that we are trying to serve directly, those who design buildings.

ii. the information produced by the research communities has been unbalanced. The attention given to physical issues - the energy consumption and environmental performance of buildings - has not been matched by equal regard for social factors - their occupants' expectations, priorities and preferences, and how they use buildings. So we know too little about the people that we are attempting to serve indirectly, those who use or occupy buildings".

Many architects echo this opinion. They accuse the researchers' work as being irrelevant to their needs (Courtney, 1989).

2.8.3 Problems with knowledge dissemination

Knowledge dissemination is the process of distributing research findings to the targeted audience(s). It is an important link in the knowledge transfer process, unfortunately, it

poses the most severe barrier at present. Richardson (1990) identified some of the common problems:

- (1) The lack of responsibility among research professionals for knowledge dissemination. At least many researchers do not have a sense of responsibility for disseminating their research results actively to those who need them.
- (2) Low priority assigned to the dissemination by researchers who usually feel that they do not have enough time and resource for dissemination. In addition they consider that they will get little credit for doing so. In the research community, publication in peer-referred journals is the normal route to promotion, while dissemination to practitioners is not highly valued.
- (3) For those researchers who do consider dissemination important, the lack of skill in this respect is a further handicap. They quite often do not know how to get to the right audiences and how to communicate with them effectively.
- (4) Most of the journals published today in the building industry are quite specialised, and confined to a very specific profession such as lighting experts, energy specialists, builders or architects. Materials aimed at cross disciplinary audiences are often difficult to get published since they often considered as inappropriate for the professional journals.

2.8.4 Problems with knowledge utilisation

Just as architects accuse researchers for their work as being irrelevant and unhelpful, researchers usually blame architects for not making sufficient efforts to make use of the research achievements. Several factors also contribute to these conflicting claims:

- (1) The limited ability shown by the architects in absorbing new knowledge, because of the lack of technical background in their training (Lawson, 1980).
- (2) The lack of clear messages about the benefits of the new knowledge and good examples in practice. The fear of failure in exploiting new knowledge forces many architects to stick to their traditional, familiar practice approaches (Brown, 1988).
- (3) The way architects work, that is mainly depending on previous experience rather than searching for new technical information (Marvin, 1985c).

2.9 Improving the Technology Transfer Process

The influencing factors and current problems in the technology transfer process have been discussed so far. There is a clear need to improve the situation and a number of actors can be identified who have a responsibility to ensure that this task is carried out effectively.

- The funding bodies.
- The research community.

- The education institutions.
- The intermediary bodies.
- The design profession.

Improvement in the technology transfer process depends upon the combined efforts of all these parties.

2.9.1 The funding bodies

Research funding bodies, such as national governments and the EEC Commission, have a strategic influence on technology transfer in the building industry. The guiding policies concerning research and application of these authorities are usually sound. For example, SERC seeks a balance in its activities of 40% long term fundamental research and 60% short term research which can yield benefits within five years (Cooper, 1989). However, the lack of effective administrative measures often results in the unsatisfactory implementation of this policy (Hall, 1989). Further efforts are needed from the sponsors, for example, supporting technology transfer oriented projects and improving the administration of knowledge dissemination across all of their research. There are some encouraging signs from the research funding bodies in recent years. Some projects, such as the ESTU Passive Solar Design Studies and the BRECSU Best Practice Programme, already put the emphasis on the practical application of existing technology (DoE, 1989).

In addition, the government could use its economic leverage and market forces to promote the utilisation of up-to-date knowledge in the building industry. For example higher fuel prices or taxation on energy consumption would encourage the application of energy conservation technology.

2.9.2 The research community

As the major knowledge producer, the research community has a special responsibility towards technology transfer. There are many areas in which the research community could make additional efforts.

- (1) First, researchers need to take technology transfer seriously. Knowledge dissemination, especially towards practitioners, should be an integrated part of the research activities where the subject has direct practical applications.
- (2) More research effort should be put into the area of the architects' knowledge requirements to build upon past research initiatives in this area (Goodey 1971, Kealy 1988).

- (3) In the light of the inappropriateness of some of the existing design support measures, greater research should be devoted to the development of suitable design support systems for architects in practice.
- (4) Section 2.6 stressed the potential importance of the application of computing in building practice as technology transfer media. Research into the development of computer based design support system has, and will probably continue to have a high profile. It will be argued, in chapter 5, that this area is one of the key areas for the future improvement of architectural design support and technology transfer in the building industry.

2.9.3 The educational institutions

The objective of the education activity for both architectural students and continuing education for architects in practice should include:

- (1) The promotion of the architects' and future architects' understanding of building technology.
- (2) The awareness of the benefits that up-to-date knowledge can bring to the building industry.
- (3) The improvements in their ability to access the new technology by equipping them with modern IT knowledge and skills.

2.9.4 The intermediary bodies

Due to the discipline gap between researchers and architects, there is a natural communication barrier between them. One solution to the problem is the establishment of intermediary bodies, such as advisory services. These intermediary bodies could then translate the problems of the industry into research programs and translate research results into information that is of practical use to the industry. One example is the Energy Design Advisory Service (EDAS) operated by the Royal Incorporation of Architects in Scotland Steering Board (Maver, 1989).

2.9.5 The design profession

The responsibility of the building design professions in the technology transfer process will not be discussed in detail. This, by no means, denies their importance in this process. The reason for not discussing their role further is because they are at the receiving end of the transfer process. This study focuses on the technology creating and sending side. The emphasis is on the understanding of designers' information requirements and the ways in which their requirements can be met in order to promote the use of existing knowledge.

2.10 Summary

This chapter has reviewed the current situation of technology transfer in the building industry. The discussions include the need for technology transfer, technology transfer media, some existing barriers as well as suggestions for improvement in general terms.

Nowadays, building practice is a highly complex process. Increasing demands on building functions and performances require specialised professional research. These tasks are generally beyond the skills of people who are directly involved in building practice. Against this background, many independent building research bodies have been established world-wide during the past few decades. These research bodies undertake studies concerning every aspect of the building environment, i.e., thermal, energy, lighting, acoustic, economic and so on. While the research community has been successful in conducting research, the transformation of research achievements into practical application has become a problem. The current gap between building research and practice has been widely acknowledged.

The discussion in this chapter has identify architects as being one of the most influential actors involved in the technology transfer process in the building industry. The next chapter will concentrate on the discussion of issues concerning architectural design and architects' information requirements.

CHAPTER 3

ARCHITECTURAL DESIGN AND INFORMATION REQUIREMENTS

3.1 Introduction

In chapter 2, it was argued that the architect profession is one of the major actors in the technology transfer process and that architectural design has a vital impact on building performance. In order to understand the complexity of architectural design many fundamental questions need to be addressed, for example,

- What is the nature of architectural design?
- What is the design process model?
- How do designers work in practice?
- What are designers' information requirements?

It will be argued, in this chapter, that the understanding of these problems is essential to the provision of appropriate design support and, ultimately, the improvement of the technology transfer process.

3.2 The Nature of Design

Design is a widely used word. It describes a creative decision making process which applies to many fields such as engineering, industrial design, graphics, painting architecture and so on. Does the word 'design' have the same meaning in all these fields?

3.2.1 Definition of design

Since the early 1960s, there has been a continuous series of study on the nature of design. Many researchers from various fields looked into this issue from different view points. Jones (1976) summarised a number of definitions of design offered by a number of authors.

"Finding the right physical components of a physical structure. (Alexander)

A goal-directed problem-solving activity. (Archer)

*Decision making, in the face of uncertainty, with high penalties for error.
(Asimow)*

Simulating what we want to make (or do) before we make (or do) it as many times as may be necessary to feel confident in the final result. (Booker)

The conditioning factor for those parts of the product which may come into contact with people. (Farr)

Engineering design is the use of scientific principles, technical information and imagination in the definition of a mechanical structure, machine or system to perform efficiency. (Fielden)

Relating product with situation to give satisfaction. (Gregory)

The performing of a very complicated act of faith. (Jones)

The optimum solution to the sum of true needs of particular set of circumstances.

(Matchett)

The imaginative jump from present facts to future possibilities. (Page)

A creative activity - it involves bringing into being something new and useful that has not existed previously. (Reswick)"

The diversity of these definitions shows the complexity of the nature of design. People from different professions gave definitions using their own vocabularies and stress the features they believe as being important. Even within the same field people's views on design change with time. For example today's architectural design is considered quite different from that of medieval times, not only in terms of the final product but also in terms of design approach.

Despite all these differences, some authors, such as Jones, believed that some features are common to all design activities. Based on an analysis of previous studies of the nature of design, he proposed a general definition for design, "designing as the initiation of change in man-made things" (Jones, 1976). He suggested that this definition applies not only to the work of engineers, architects and other design professionals but also to the activities of economic planners, legislators, managers, publicists, politicians who are in the business of markets, public services, laws and the like.

Similarly, Evans (1982) summarised the position with the observation that "any act of changing the world is an act of design, not just machine-tools, furniture or house design, but also the development of health care system or legislation; all man-made changes affecting our lives are by design".

However, on the other hand, some authors doubt the value for having a universal definition for design and have instead highlighted the differences in various design activities. For example, Lawson (1980), in his book *How Designers Think*, began with the explanation of the complexity of the designers' role in society rather than the definition of design. He argued that

"to attempt a definition of design too soon might easily lead to a narrow and restricted view. To understand fully the nature of design, it is necessary not only to seek out the similarities between different design situations but also to recognise the very real differences."

In his opinion the important thing is the real implication of design in each specific design field. An over generalised definition such as Jones' "the initiation of change in man-made things" does not help to understand the nature of design. Studying the nature of design is

to explore what is involved in the design process, and not seeking a universal definition, an opinion shared by a number of authors in recent studies, such as Heath (1984), Broadbent (1988), Cross (1989).

3.2.2 Design form and context

Despite of the disagreement on the definition, there is a general agreement on the existence of form and context in a design activity. Alexander (1964) made the following classical observation on this issue:

"When we speak of design the real object of discussion is not the form alone, but the comprising of the form and its context. ... The form is a part of the world over which we have control and which we decide to shape while leaving the rest of the world as it is. The context is that part of the world which puts demands on this form; anything in the world that makes demands of the form is context." He went on to explain his view on design as searching for the good fitness between the form and context. *"Fitness is a relation of mutual acceptability between these two. In a problem of design we want to satisfy the mutual demands which the two make on one another".*

Unfortunately, as Alexander acknowledged, in most design situations, the contexts are too complex to be expressed in a unitary fashion. There are many elements in a design context and these elements do not always have equal importance. Further more, conflicting requirements are often caused by the different elements. Therefore, it is impossible to define a standard for 'good fitness' between a form and its context. In practice, a 'good fitness' or 'misfit' is to a great extent a matter of subjective judgement by the designer.

In a real world context, there is no one-to-one mapping between design context and its form. In other words, a number of forms can satisfy the requirements in the same context. Designers have a certain freedom to choose the form which they consider to be the best as long as it satisfies the context's demands. Similarly, it should be possible to apply the same design form to different contexts.

The relationship between context and form is not a simple one as cause and result. In many cases, they have an interactive impact upon each other. Taking chair design as an example, the American architect Philip Johnson is reported to have observed that some people find chairs beautiful to look at because they are comfortable to sit in, while others find chairs comfortable to sit in because they are beautiful to look at (Lawson, 1980).

Another feature of design is that the design context does not offer a multiple choice of the possible forms. On the contrary, designers have to study the context and propose solutions

of form accordingly. The complexity of design contexts is also acknowledged by Cross (1989). He pointed out that the "problems designers tackle are regarded as 'ill-defined' or 'ill-structured', in contrast to well-defined or well-structured problems such as chess-playing, crossword puzzles or standard calculations". The characteristics of the design problems have been summarised by Cross (1989) as follows:

- (1) There is no definitive formation of the problem.
- (2) Any problem formulation may embody inconsistencies.
- (3) Proposing a solution is a means of understanding the problem.
- (4) There is no definitive solution to the problem.

3.2.3 Art, science and design

Traditionally, design is often related to art, but recently some people have tended to associate design with science (Jones, 1976). However, is design a kind of art, or is it a discipline of science? Understanding the difference and common characteristic between art, science and design is an aid to understand the nature of design itself.

There is no doubt that designers' work involves an element of art. Jones (1976) pointed out that

"the artistic approach is relevant when designers have to find their way through a vast number of alternatives while searching for a new and consistent pattern upon which to base their decisions."

Lawson (1980) also claimed that "it is sometimes difficult to separate design from art". However, both Jones and Lawson acknowledged the differences between design, art and science.

"The main point of difference is that of timing. Both artists and scientists operate on the physical world as it exist in the present (whether it is real or symbolic). ... Designers, on the other hand, are forever bound to treat as real that which exists only in an imagined future and have to specify ways in which the foreseen thing can be made to exist" (Jones, 1976).

Jones explained the differences among art, science and design in terms of what needs to be addressed. Lawson (1980) gave a similar argument.

"What is usually different however is the nature of the source of the problem. Perhaps it is only a difference of degree, but nevertheless the difference is real enough and of fundamental significance in determining the nature of the process of art and design".

A consequence of these differences is that the artist can deal with issues and solve

problems which seem important to him. An artist is usually responsible for his own work and is free to shift his attention and change the problem to one which fascinates him more.

However, "In this respect the designer is faced with a different situation altogether. In design, the problem usually originates not in designer's mind but with a client or user; sometimes who is unable to solve the problem, or perhaps, even fully to understand it without help. The designer, unlike the artist, is almost always commissioned; the task, albeit ill-defined, is brought to him" (Lawson, 1980).

Unlike artists, designers generally work within a much stricter boundary, they cannot devote themselves exclusively to problems which are of most interest to them personally. Their freedom of creation has more constraints. They have to meet the requirements of the clients and the work within the relevant legislation and other external constraints.

The value of a work of art comes mainly from the appealing aspects of its form; while in design, satisfying the functional requirements is usually the most important consideration. Again in the example of chair design mentioned above, no matter how beautiful it looks, it is not a good chair if it is uncomfortable for people to sit in. Therefore, to meet the need of people sitting on comfortably should be the top priority in chair design with possible exceptions for exhibition purposes.

3.2.4 Creativity and rationality

Creativity and rationality are two outstanding features of human beings. Creativity enables people to initiate ideas while rationality allows them to pursue ideas logically based on given circumstances.

Creativity is essential in almost all human activities, Broadbent (1973) offered the following observation on creativity in art, science and design (table 3.1).

Table 3.1 Creativity in art, science and design (after Broadbent, 1973)

ART	To paint an 'original' picture, write an 'original' poem, piece of music and so on.
SCIENCE	To formulate a new theory or postulate a new hypothesis.
DESIGN	To solve a technical problem in a new and more 'elegant' way. Elegant in this sense means simple, efficient, economical, etc.

The different implications of creativity in these fields are obvious. To paint an 'original'

picture is a process driven by the painter's subjective desire. Two painters could produce totally different pictures on the same subject. To formulate a new theory in science is to find the way to explain a natural phenomenon. It is a process driven very much by objective rationality while requiring the creative thinking of the scientist. The solution to a scientific problem is usually unique.

Design, such as designing a car or building, lies between the artistic creation and scientific discovery. It offers scope for designers' subjective creativity on one hand. On the other hand, many constraints of design are often beyond designers' control. Designers can not alter them but have to work under the given constraints. In this sense, there is a strong rationality in the design process. The design process demands a good balance between creativity and rationality.

The distinction between art, science and design is reflected by the degree of balance between creativity and rationality. While work in all these three areas demands both creativity and rationality, the major value of art is its creativity; the major feature of science is its rationality. Design is somewhere between these two (figure 3.1). It could be argued that a process without either creativity or rationality is not a process of design.

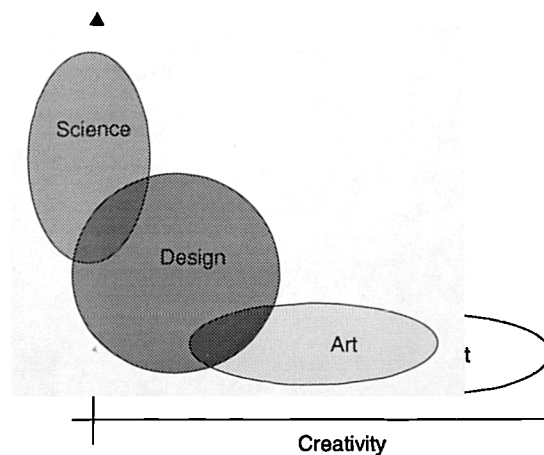


Figure 3.1 The balance between creativity and rationality

3.2.5 Conclusion

The following conclusions can be summarised from the above discussion.

- (1) Design is a creating process within a specified context for a pre-defined purpose.
- (2) Most of the design contexts are 'ill-structured', in other words, there is no definitive formulation of a design problem and there is no unique solution to a design problem.
- (3) Design differs from both art and science, it requires an even balance between

creativity and rationality.

The objective of studies on the nature of design is

"to make public the hitherto private thinking of designers, to externalise the design process. ... the underlying aim is to bring designing into open so that other people can see what is going on and contribute to it information and insights that are outside the individual designer's knowledge and experience"
(Jones, 1970).

Proposing a definition of design is unlikely to achieve this objective. Instead, it requires an in-depth analysis of the design activity and the design process.

3.3 Design Process Models

Design is a decision making process and decision making is part of people's every day life. Decisions are made when people are shopping, travelling, eating, teaching, doing research, designing, etc. Some decision makings are self-conscious others are not. Nevertheless, it can be argued that they all follow a similar pattern at a general level (figure 3.2).

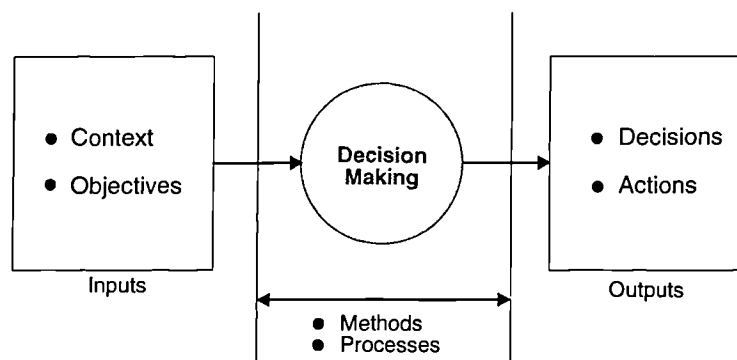


Figure 3.2 Decision making pattern

The context and objectives define the inputs to decision making. The decisions taken and the actions followed are the results of the decision making process. Despite the similarity of the decision making process, there are fundamental differences depending upon the decision making context and objectives. For example, when people buy clothes, they usually have fairly clear objectives of what they want. They find out what is available to meet their needs from a finite collection in the shops. Based on availability and the balance of price, quality and other factors, decisions are made to purchase a particular product. This process is relatively open and can be explained.

However, in the case of building design, the situation is different. The existing conditions concerning a building project cannot normally be exclusively listed out. The objective and brief are also difficult to define at the beginning. Decision making in design is not to select from a fixed set of options but to propose and generate solutions. The difference in the nature of the problem between buying clothes and designing a building determines that the decision making processes are markedly different in these two cases.

The aim of the following analysis is to understand the design process in order to integrate technical support into the architectural design process. Developing a design process model or design methodology is not the aim of this study and the analysis is focused on existing studies in this area.

Design method and design process, though not synonyms, have very close meanings.

- Design method represents an identifiable way of working in the context of design.
- Design process describes the sequence according to which the design work is carried out.

Since this study is more interested in the sequence of how the design decisions are made, design process is the focus in the following analysis. Many models have been proposed by different authors in previous studies to describe the design process, which can be broadly divided into three types:

- normal linear process,
- iterative linear process,
- iterative cyclic process.

The characteristics and some examples of each type are discussed in the following.

3.3.1 Normal linear process

With the linear nature of the time dimension, events always happen in a linear pattern in the real world. Therefore, it is not surprising that the earliest design process models were linear emphasising the linearity of the design process.

The most general linear model describes the design process as consisting four steps, **brief**, **analysis**, **synthesis** and **evaluation**. According to this model, the designer first gets an overview of the problem in a typical design situation which is the **brief**, then proceeds to,

"Analysis in which all the design requirements are listed and reduced to a set of logically related performance specifications.

Synthesis in which solutions are found for individual performance specification and then built up to form a complete design.

Evaluation in which alternative designs are tested against performance

specifications--particularly those concerned with operation, manufacture and sales." (Broadbent, 1973)

This general model was adopted by many people from different disciplines with some variances, such as scientific research, medical diagnosis, business management. For example, Gugelot (1963), from the point view of industrial design, suggested the following process model.

- (1) Information stage: find out as much relevant information as possible.
- (2) Research stage: study the design context.
- (3) Design phase: find new formal possibilities to fit design context.
- (4) Decision stage: seek a favourable decision from sales and production management.
- (5) Calculation: adjust the design to specific production standards.
- (6) Model-making: build a prototype, a working model to demonstrate the limits of any technical risk involved.

Mapping to the model of brief-analysis-synthesis-evaluation, the information stage is the briefing; research stage is the analysis; design and decision are synthesis; and calculation is the process of evaluation. This model is effectively a version of the general linear design process model, which was applied specifically to industrial design.

The RIBA's plan of work describes the building process in twelve stages as shown in figure 3.3:

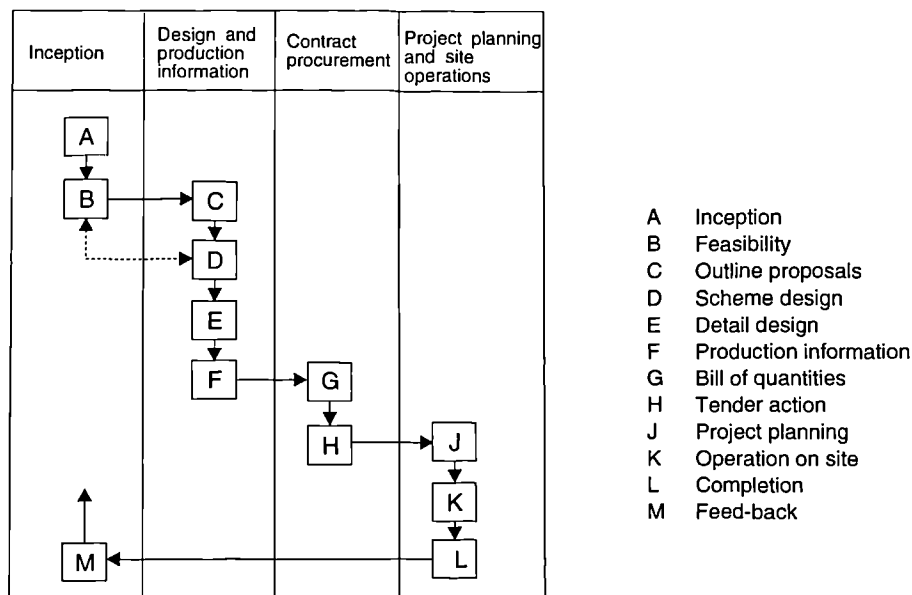


Figure 3.3 RIBA's plan of work (after RIBA, 1988)

The RIBA's plan of work is essentially a linear procedure, with limited cyclic refining

actions at the early design stages, indicated by dotted line in Figure 3.3. According to this plan, aspects of a design need to be 'frozen' in increasing detail at each stage. Subsequent changes are to be avoided, otherwise, they may result delays and extra costs. However, in practice, architectural design is far more complex than what this figure suggests. Many decisions have to be refined continuously and they can not be 'frozen'. In addition, the RIBA's plan of work is a description of the results of building process rather than the process itself. It tells "what architects must do rather than how it is done" (Lawson, 1980).

Maver (1970) also tried to model the architectural design process. He suggested that an architect needs to go through the sequence of analysis, synthesis, appraisal and decision at increasingly detailed levels during a design (figure 3.4).

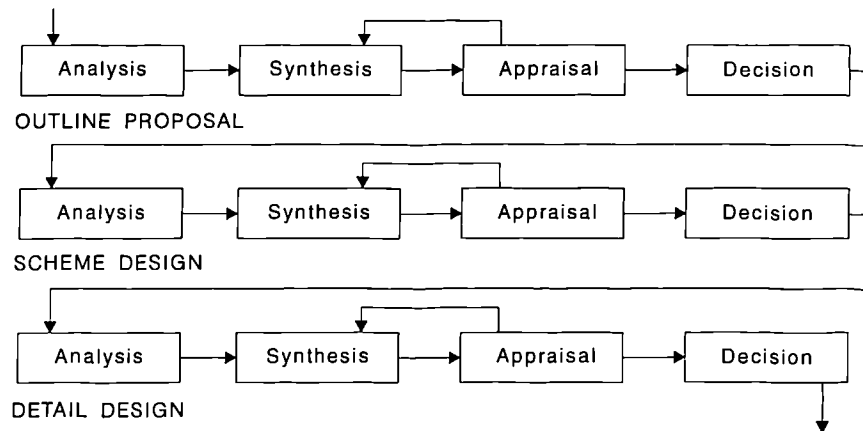


Figure 3.4 The Maven's map of the design process (after Maver, 1970)

The design process models introduced above are all simple linear process. According to these models, design decisions are made in a sequential manner. One must complete one procedure before moving on to the next stage. For example, in Gugelot's model, Information and Research stage should be completed before Design phase begins. In RIBA's plan of work, detail design cannot start before the completion of scheme design. Later decisions are made based on the assumption that the previous ones are complete, little backward rethink is expected in these design process models.

3.3.2 Iterative linear process

However, many authors have argued that some of the assumptions underlying the normal linear process model are not always valid (Page 1963, Lawson 1980, Broadbent 1988). They proposed iterative linear models of the design process.

While agreeing on the basic analysis-synthesis-evaluation pattern of decision making, Page

(1963) had argued that

"in the majority of practical design situation, by the time you have produced this and found out that and made a synthesis, you realise you have forgotten to analyse something else, and you have to go round the cycle and produce a modified synthesis, and so on".

This suggests that simple linear process models miss the backward thinking when new aspects are discovered about the design problem. Therefore, Page suggested that there should be return loops from each stage to all the previous stages. Subsequently, the simple linear decision making process has been developed into an iterative linear process as shown in figure 3.5.

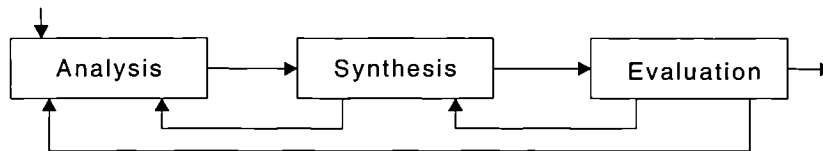


Figure 3.5 Page's decision sequence model

Lockley (1987) presented a modified linear design process model of the RIBA's plan of work. Instead of describing design as a one directional process from briefing to building completion, Lockley proposed return loops to previous stages in the light of new information (figure 3.6). He believed that these return loops introduce inefficiency into the design work. He implied that, in an ideal situation, designers should gather enough information before making related decisions in order to avoid these inefficient return loops.

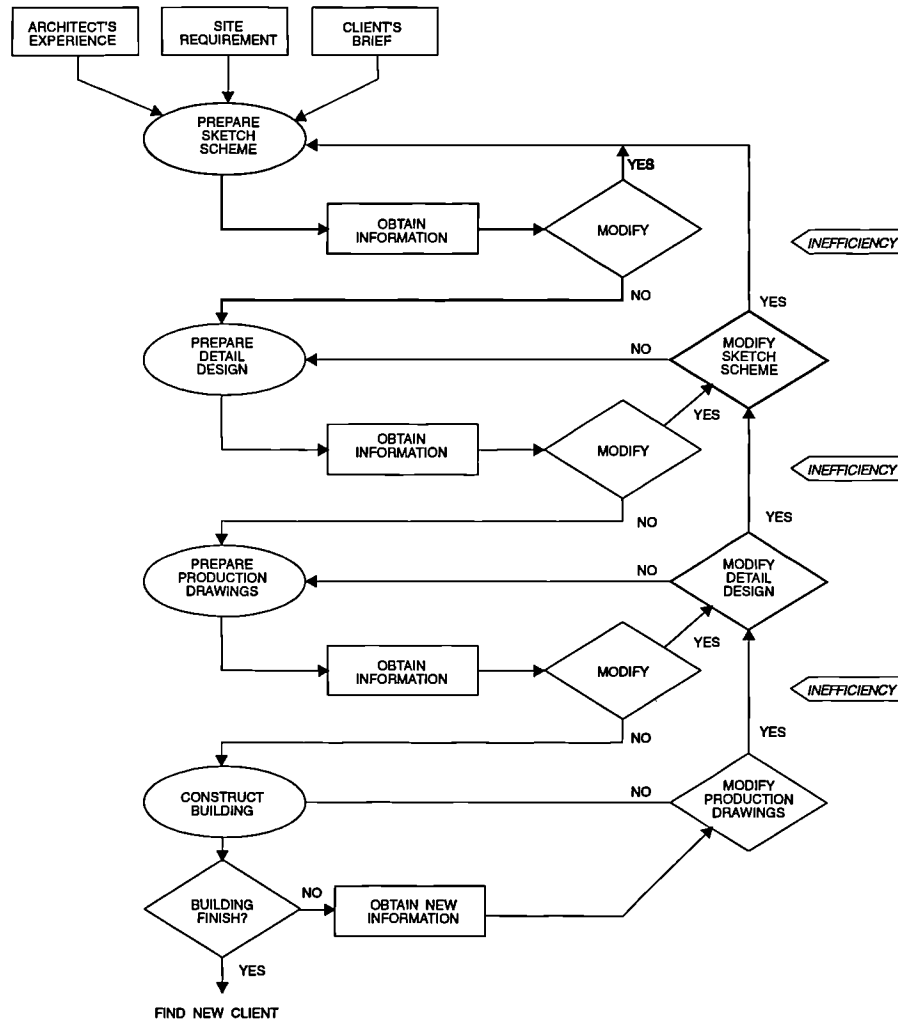


Figure 3.6 Lockley's map of design process (after Lockley, 1987)

This argument, however, may not disclose the real reason for the frequent returning loops in the building design process. There might be an inevitable logic behind this kind of design practice; designers might simply have different working styles from other professions as suggested by Lawson (1980).

The design process models discussed so far, including the simple linear ones and the iterative linear ones, all start with the analysis of the design problem and end with the solution to the problem. This analysis-synthesis-evaluation approach of problem solving is usually described as the problem-oriented strategy.

There is another approach, which is commonly known as "trial and error". According to this alternative approach, design decisions are initially made on the basis of limited information. Later on the decisions are refined or changed in the light of more information generated in the process of investigating the proposed solution. Since it is centred on the trial of different solutions in order to find the most appropriate one, this approach is also

known as the solution-oriented strategy.

The difference between these two approaches was exemplified in a study by Lawson (1980). He conducted an experiment to compare the problem-solving strategies between architects and scientists. The experiment was carried out by architectural and science postgraduate students. The task was to arrange modular coloured wooden blocks onto a four by three bays rectangular plan, a simplified design problem.

As a result, he found that quite consistent and strikingly different strategies were used by the two groups of participants.

"The scientists adopted a technique of trying out a series of designs which used as many different blocks and combinations of blocks as possible as quickly as possible. Thus, they tried to maximise the information available to them about the problem. If they could discover the rule governing which combinations of blocks were allowed they could then search for an arrangement which would optimise the required colour around the design. The architects on the other hand selected their blocks in a quite different way. The eight blocks were examined to see which four blocks had the most of the desired colour, and the first design was built from these four blocks. If this proved not to be an acceptable combination then the next most favourably coloured block would be substituted and so on until an acceptable solution was discovered." (Lawson, 1980)

It is obvious, in this case, that the scientists adopted a problem-oriented strategy while architects had a solution-oriented strategy. The architects were obsessed with achieving the desired result while paying little attention to the rules of the problem. They ignored examining alternative solutions after finding the desired one.

In study of design decision-making in architectural practice, Mackinder and Marvin (1982) had similar findings. They discovered that architects intended to create a solution to a design problem as quickly as possible at the very beginning by using their experience without looking for additional information. As the work proceeded, more problems and the inappropriateness of the solution were discovered and the solution was improved gradually. For example, it is very common that when an architect designs a hotel, he would not start from absolutely scratch. Instead he would have an image or images of the hotel in his mind based on his previous educational or professional experiences. In design the imagined solutions will be specified in the light of design requirements and constraints.

Darke (1978) also found this tendency of architects adopting a solution-oriented problem solving strategy. He described this working style as generator-conjecture-analysis (figure 3.7). In other words, the architects tend to develop a quick solution on the basis of their

understanding of the problem, then examine the solution in further detail and try to discover more of the problem and refine the solution. The whole design process proceeds in such a pattern at increasing levels of detail.

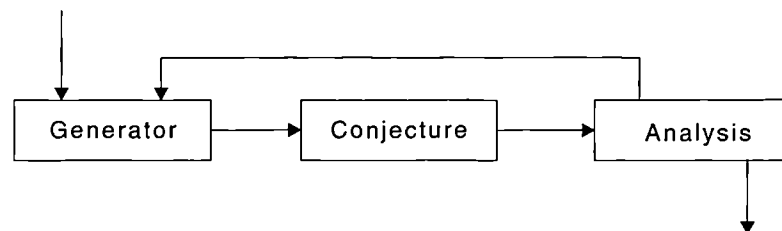


Figure 3.7 Darke's map of the design process (after Darke, 1978)

The discussion in section 3.2 of design nature identified two important elements of design, creativity and rationality. In term of problem solving techniques, these two aspects have different requirements. Rationality requires a problem-oriented problem solving strategy. All aspects of the problem are analysed thoroughly before the solution can be developed rationally. On the other hand, creativity needs a solution-oriented strategy. Under some circumstances, it is not practical to spend too much effort analysing all aspects of the design problem. Instead, the solution-oriented strategy of generating and evaluating quick solutions may be more appropriate.

It is highly possible that designers, consciously or unconsciously, use both of these problem-solving strategies in most design situations. Sometimes they may start with the analysis of the problem and obtain the solution rationally. In other cases, they may start the design by trying alternative solutions and see which is the best fit. Therefore even an iterative linear model cannot represent such complexity.

3.3.3 Cyclic iterative process

The linearity of the process depends on being as nearly right as possible at each step because the opportunities to return to correct errors are seen as a liberty or inefficiency rather than a natural part of the process (Hickling, 1982).

However, it is impossible to prove anything as being right, in fact we can only learn by being wrong (Popper, 1968). "Learning from mistakes" and "Trial and error" which are common wisdom which exists in nearly all cultures also applies to the design process. Therefore, an ideal design process should be one which promotes the opportunity to be wrong, with little or no penalty for being so. Pursuing this line of reasoning, Hickling suggested a cyclic, iterative and whirling model (figure 3.8), which "allows the input of creative thought free run".

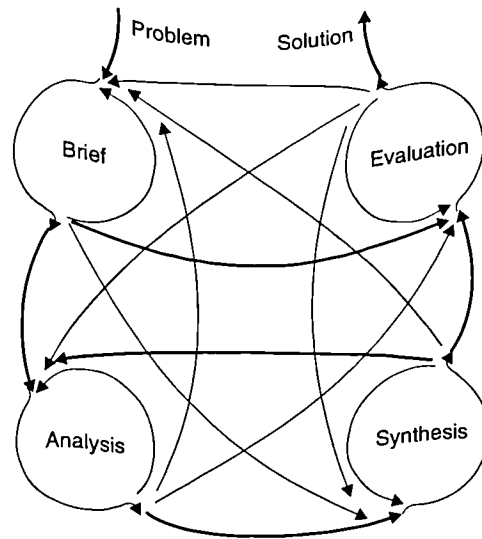


Figure 3.8 The cyclic iterative process (after Hickling, 1982)

In the cyclic process model, there is greater flexibility to skip forwards and backwards. Linear process models describe the design process as brief-analysis-synthesis-evaluation or brief-synthesis-evaluation-analysis with or without return loops during the process. The cyclic process model accommodates all the linear process attributes and a lot more. For example, it is possible to go from analysis to any of the other activities. In addition, it is possible to enter the analysis from any of the other activities. This model covers both problem-oriented and solution-oriented problem solving approach discussed above. It has "more complexity", "less simplification" and is closer to "what really happens" (Hickling, 1982).

Real world problem solving is much more complex than a single layered brief, analysis, evaluation and synthesis. Very often each of the four activities is a decision making process itself, and the cyclic decision making process model applies to it. Hickling proposed an extended cyclic process model and gave an example (figure 3.9). He pointed out that this model could be extended even further to several levels.

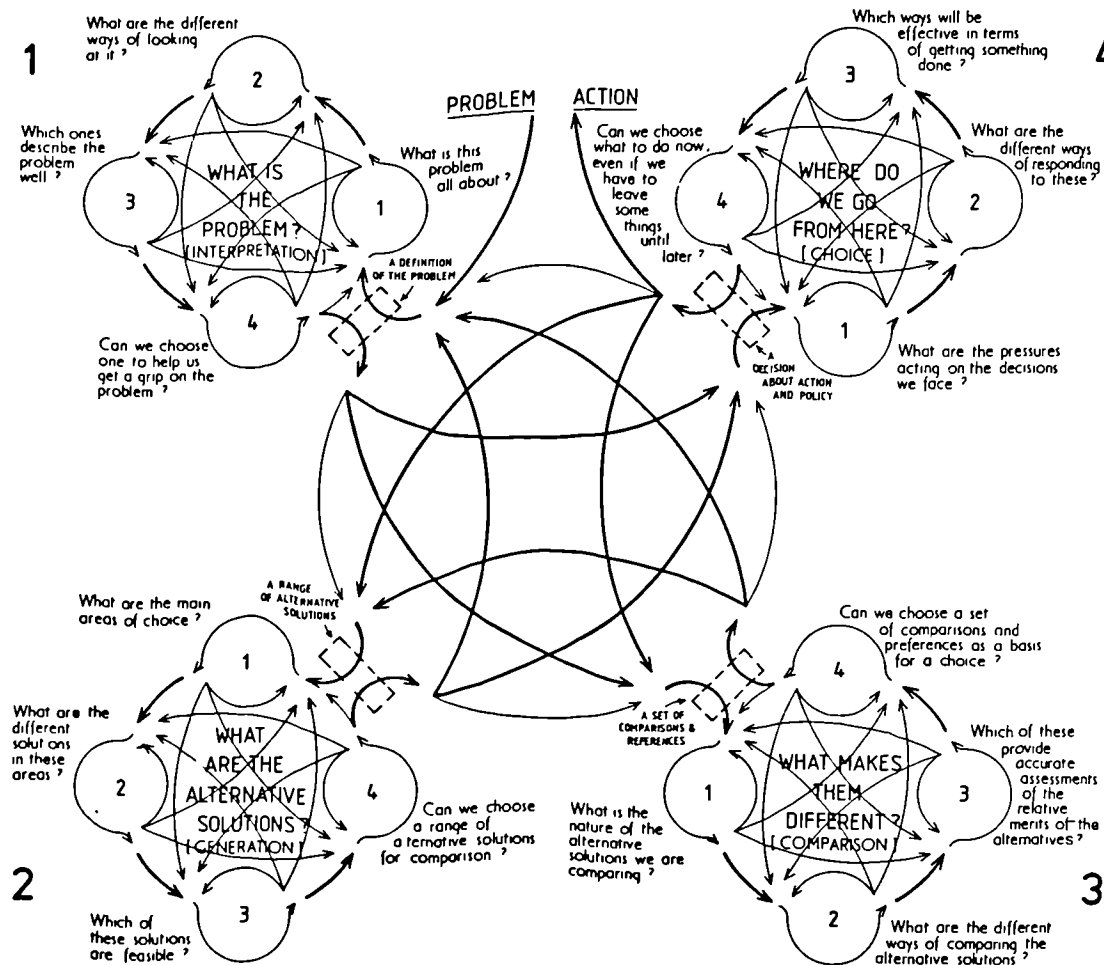


Figure 3.9 Cyclic process model for problem solving (after Hickling, 1982)

3.3.4 Characteristics of the design process

It has to be acknowledged that there is no general agreement on the design process model. Nevertheless, some important conclusions can be drawn from the analysis of this section, which have implications for the provision of architectural design support and technology transfer in the building industry.

- (1) Design is an endless process. The solution to most design problems is not unique. The searching for alternative solutions can always go on in design. The end of the design process is usually not marked by finding the ultimate answer to the problem. Rather, "the designer identifies the end of his process as a matter of judgement. It no longer seems worth the effort of going further because the chances of significantly improving on the solution seem small." (Lawson, 1980).
- (2) Design is an iterative and interactive process. Design is a process of finding problems as well as solving problems. Not all aspects of the problem can be listed at the beginning of a design; some emerge during the design process. Therefore it is

necessary to go back to re-examine previous decisions. Design is not a one directional process; return loops are necessities.

- (3) Design is a cyclic process. Design decisions are not linear or hierarchy arranged. It is not always the case that some decisions are bound to be made after others. For example, some more detailed decisions go before general ones and vice versa.
- (4) Design is a compromise process. Unlike scientists, designers cannot always find the unique right solution at the end of design. Their solution is always an optimisation to a particular situation based on their own judgement. On the other hand, unlike artists, designers do not have the freedom to do what they like to do. They have various limitations from client's requirements to site constraints. They do not necessarily satisfy all aspects of the design problem with the final solution. In stead, they usually do the best they can under a particular set of conditions.
- (5) There are two decision making strategies, problem-oriented and solution-oriented which are often used in the design process.

The above remarks are made with special reference to the architectural design, it is suggested that they also apply, to a large extent, to other design activities.

Creating a design process model is to present a simplified version of the reality of what is happening in design. This simplification is necessary in order to understand the complex world and communicate that understanding. However, if the process model is over-simplified, some of the important aspects of the design process might be excluded. Consequently, the model would become invalid in some cases. On the other hand, if the complexity of the model increases indefinitely, it will end up as a replication of the real world, in which case, it would not help to understand the design problem and could not achieve the objective for having a model in the first place. Formalising a design process model is a matter of striking a balance between simplicity and reality.

Adding to the difficulty of developing a design process model is the fact that it is always difficult to study what happens in the designers' mind. In the case of architectural design, first it is almost impossible to monitor designers at work without disturbing them and distorting the observations. Second, results of the observation are results of designers' work, the drawings, reports, etc. What we need to know is how designers get these results, in other words, the process of thinking and decision making within the designer's mind.

3.4 Architectural Design Practice

The previous sections have reviewed the theoretical literature relating to the nature of design and existing studies on the design process model. Unanimous agreement does not exist in either of these two areas. This section examines to what extent these process

models can be mapped to the practice of architectural design. In addition to the observation of design practice, issues of interests in this section include how do designers use technical information during their work? How do they learn? and so on.

3.4.1 How do designers work?

The RIBA's plan of work is a general guide for architectural design practice in the UK. However, it specifies the production schedule of the design work, rather than the actual sequence of the designers' thinking. In practice, the design decision making process depends very much on the individual designer and the design project. Nevertheless, the RIBA Plan of work is widely used in architectural design practice especially in project management. For instance, the design progress and fee charging are normally related to the Plan of work. Therefore, in the following discussion, special reference is kept to the Plan.

A series of research studies on architectural design practice was carried out at the Institute of Advanced Architectural Studies, University of York. The main aim of the research was to examine the roles of information, experience and other influences during the design process through observations of architects at work (Mackinder, 1982). Six architectural design offices were chosen for case studies. Eleven design projects in these design offices were followed. The method of the study included interviewing architects and analysing architects' diaries completed during the progress of the work.

This research has identified specific features of the architects working style, which serves to explain some of the current difficulties in the provision of technical support during the design process. Their main findings were as follows:

- A design team for one project usually consists of several members of staff. There is frequent communication among the staff as well as other relevant participants, i.e., clients or contractors. Many design decisions are made through personal contacts.
- Designers usually shift frequently between several predominant activities, such as (i). drawing, (ii). preparing reports and correspondence, (iii). meeting and discussions. They do not have time to read lengthy technical information.
- The outline design concept is usually produced by an experienced architect. Alternative concepts are rarely explored because of time pressure during this stage. More detailed work is usually done by less experienced members of staff. This makes refining design ideas a difficult task.
- Special consultants are only consulted for technical issues, such as structural calculations.
- Architects tend to rely on their existing experiences or personal contacts rather than

published information when tackling design problems. This poses many difficulties for the provision of published design support materials.

- Most designers work on several projects concurrently; they often work under heavy time pressure. In most cases, their objective is to get the job done, not to explore other alternatives.
- Architects do not record the progress of their work in great detail. They trust their memories to ensure that they will not go through the same 'loop' of decision making again for the same job. While an individual designer can use previous experience in similar cases later, other members of the design team might have to repeat the same process unnecessarily due to the lack of working records within the team.
- The designers do not normally work according to a pre-structured plan. There are many factors influencing the pace of the work, which are beyond the designers' control, i.e., waiting for planning approval or clients' replies to queries.
- The design time spent on different design issues is not always in proportion to their importance. For instance, a designer might spend half a day on the layout of the house but a whole day on the selection of a carpet pattern for a room.

Mackinder (1982) summarised the architectural design process in practice as follows:

At the very beginning of a design project "an initial concept for the building plan, form and general construction was developed rapidly using little information other than the client's brief, the site constraints and the designer's own experience. This initial concept was then developed and refined, using more deliberately research information and later modified as necessary, in response to emerging constraints and changing requirements"

A survey conducted by the RIBA (1984) supports Mackinder and Marvin's conclusions. This survey examined the time spent by designers at each design stage and the decisions made. Figure 3.10 shows the result of the survey. It includes the average percentage time designers spend combined with the symbolic number of decisions made at each stage. Obviously, time spent is relatively easy to measure but not the decisions made. The decision measurement is therefore symbolic, considering both the number and importance. The implication is that designers use less time but make relatively more decisions at the beginning of a design.

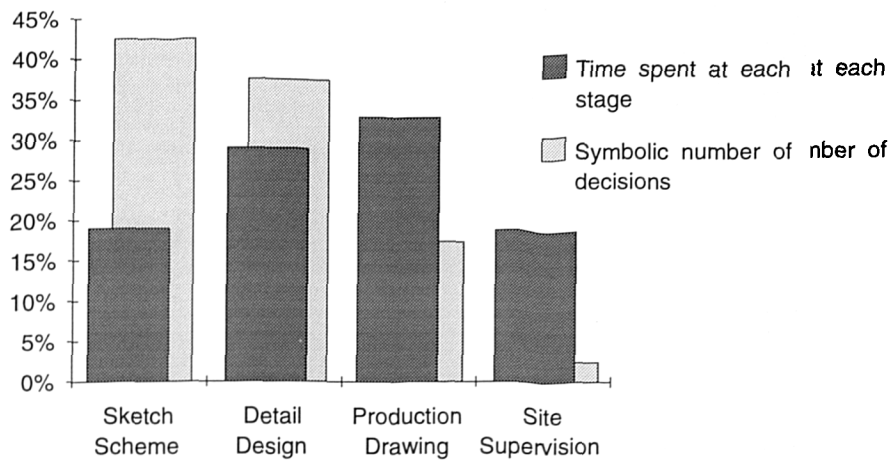


Figure 3.10 Average time devoted to design stages (after RIBA, 1984)

Mackinder and Marvin's study and the RIBA's survey confirm Lawson's (1980) argument that designers tend to pursue a solution-focused strategy in their work. They like to get a solution even before they get the entire problem clear. Therefore, architectural design is inevitably an iterative process of problem solving and problem finding, with the initial design ideas being refined and revised at later stages.

3.4.2 How is published information used?

Designers have a genuine tendency to avoid using written information as far as possible. Observations by Mackinder and Marvin (1982) revealed that experiences and personal communications are preferred information sources over written materials. Similar findings have been reported by others (Yoon 1987, Kealy 1988, Powell 1988)

In later research, Marvin (1985a) found that the use of written information by designers varied at different stages of the design. At the early stages of design including inception, feasibility and sketch design, decisions are made based mainly on previous experiences. They felt competent to work relying on their previous training, working experiences, stereotype solutions and their imaginations. During the detail design stage, more written information sources are used. At this stage, many decisions, i.e., specification of wall finishes, window types, etc., need to be taken at a detailed level. For these tasks, designers can no longer count on their memory or experiences. Consequently, a more intense use of written information was observed.

Brewer and Snow (1988b) reported a BRE survey on the information use during the design process, during which 92 designers were interviewed face-to-face. The result revealed a coherent picture shown in figure 3.11. It confirms Marvin's work that during detail design the highest proportion of technical publications were consulted.

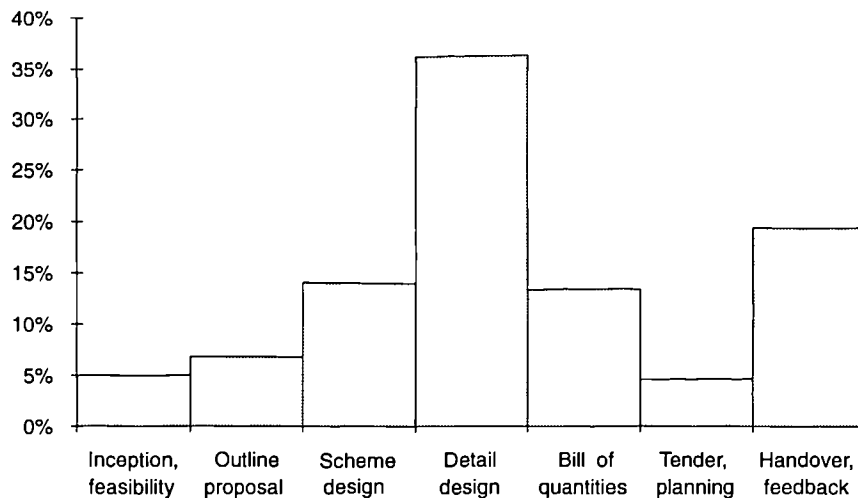


Figure 3.11 Use of technical publications during building projects (after Brewer, 1988b)

The combination of figure 3.10 and figure 3.11 illustrates the existing problem in architectural design practice. At the early design stages, designers spend less time, consult fewer information sources and make more decisions. In addition, these decisions usually have vital influences on the ultimate quality of the building. The under-use of written information at the early design stage by the designers may have been caused by the following reasons:

- (1) Designers feel competent to cope with projects they encountered, at least for domestic projects, by using their experience of previous designs.
- (2) Designers in general tend to be reluctant to spend time looking for technical information because of time pressure.
- (3) Designers often complain about the inappropriateness of the current format of technical information.
- (4) Many of the issues which have highest priority in research, i.e., energy consumption, do not necessarily have the same priority for either designers or clients.

It is argued that because many decisions are taken at the early design stage without the benefit of technical information support, they are less likely to be the most appropriate solution to the problem. The issue here is not one of designers adopting a solution-oriented strategy but rather the lack of adequate provision of information. There are two possible consequences in this scenario.

- (i) The design will proceed under the constraint of the initial design decisions, eventually leading to a design solution which may be of poor quality.
- (ii) The development at the later design stage would require an overturn of the initial design decision, consequently resulting in inefficiencies in the design process (Lockley, 1987).

Because of the importance of the early design stage and the likelihood of problems being

generated during this stage, the information requirements of designers will be further discussed in section 3.5.

3.4.3 How do designers learn?

The study by Kolb (1976) on human learning provides a basis for the analysis of designers' learning styles. He suggested four elements in the learning cycle,

- concrete experience,
- observations and reflections,
- abstract concept formation and,
- testing implications.

According to this learning cycle model, the learner has immediate experience of the existing world, steps back to reflect on this experience in context, thinks up an abstract model to explain the phenomenon, then tests the abstract model by further active exploration. People use these four processes to learn, however, not necessarily with equal emphasis on each process. Some people might favour one particular process while other people favour others. Based on this, Powell (1988) suggested different roles for these four processes, Sensor/Feeler, Watcher, Thinker and Doer as shown in figure 3.12.

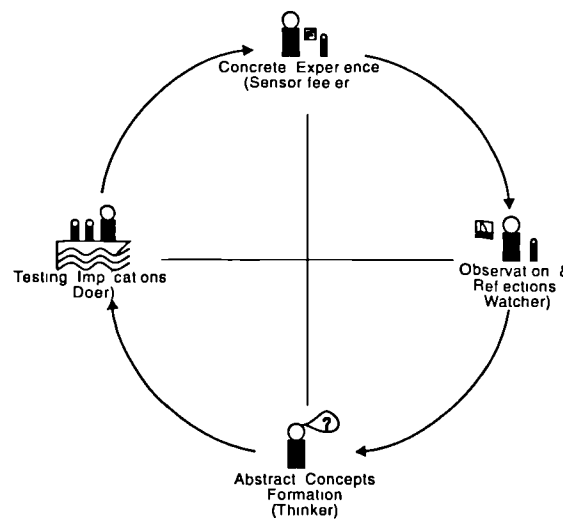


Figure 3.12 The learning cycle (after Powell, 1988)

Powell pointed out that architects usually "stick" to a preferred learning approach which corresponds to their working style. He argues that the learning approach adopted by different designers has a distinct emphasis on the four elements of the learning cycle described by Kolb. Some designers prefer to learn through watching and observing.

Others prefer thinking or practising. Powell identified four different types of architects in term of learning style. They are "Dynamic architects", "Focused architects", "Rigorous architects" and "Contemplative architects". Their respective learning styles are indicated in table 3.2.

Table 3.2 Learning techniques of architects

	Sensor/Feeler	Watcher	Thinker	Doer
Dynamic Architects	✓			✓
Focused Architects			✓	✓
Rigorous Architects		✓	✓	✓
Contemplative Architects		✓	✓	

Powell's explanations on the characteristics of the learning and working styles of each type of architect are briefly summarised as follows:

- **Dynamic Architects** have a learning approach which is centred on 'doing and sensing'. They are innovative, acting for the moment, sensing its potential and doing something about it immediately. They like to innovate personally, and like information only in short bursts, such as that in the architectural magazines.
- **Focused Architects** concentrate their attention on decisive actions. They regard themselves as 'doers' rather than 'watchers'. They would make the best use of what is available. They will tend to actively pursue information, even if it is not optimally presented. 'How to do it' is the key to their involvement in the design process.
- **Rigorous Architects** understand the world by creating abstract models. The goal of this type of architects is to become expert at what they are doing. They stand as the guardians of professional standards and aim to produce competent solutions. They want the information to be well validated.
- **Contemplative Architects** strive for wholeness, but usually have an uncertain grip of reality. They would thoroughly analyse all the data at hand in order to get a solution. They like to retrieve information sources as far as possible.

The implication of the different learning and working styles of the designers is that it may not be possible to use a single fixed way to present information to all designers. For instance, dynamic architects want the information short and brief while contemplative architects want the information to be presented comprehensively. In order to meet the requirements of a wide range of architects, the presentation of the information should be flexible and meet a variety of criteria.

3.5 Information Requirements at the Early Design Stage

Several studies on architectural design practice, discussed in the proceeding section, highlighted the problems at the early design stage. Sufficient information has to be integrated into this stage in order to improve the architectural design practice. The discussion in this section will concentrate on the designers' information requirements and the criteria that the information provision should meet.

3.5.1 Domain of the early design stage

The early design stage is defined as covering Inception, Feasibility, Outline proposals and Scheme design in RIBA's plan of work. During this stage, the major design tasks include the following (RIBA, 1988):

- Designers need to analyse client's demands and discuss with the client the development of the project to ensure that it is feasible functionally, technically and financially.
- Based on the feasibility analysis, designers make outline proposals in all aspects of the design, i.e., building layout and element construction.
- At the end of early design stage, specific solutions should have been formulated on the planning arrangement, building appearance, construction method, outline specification and cost, etc.

Despite many designers may adopt a solution oriented working strategy and they tend to make design decisions based on their experiences rather than consulting technical information, the provision of well-presented information would undoubtedly enable designers to make decisions on a better informed base, which in turn would improve the transfer of technical information from research to practice and subsequently improve the quality of building design. In other words, designers do need information support during design especially at the early design stages. This has been confirmed by Mackinder and Marvin's (1982) study and BRE's survey (Brewer, 1988b).

3.5.2 Designers' information requirement

One conclusion of the study on design process so far is that the decision making process consists of numerous steps of analysis, synthesis and evaluation in no particular order. Designers make decision based on both internal and external inputs. Internal input refers to the personal experience of the designer; external input means information which comes from outside. The common external input to design includes client requirements, existing conditions, technical requirements, legislation requirements, etc.

Personal experience is the designers' own existing knowledge and they use this unselfconsciously when needed. On the other hand, the acquisition and usage of external information require the active effort of the designers to access the information they need.

Designers use external information input for two purposes, to generate and to evaluate ideas. In this sense, the external information input could be divided into two types:

- **Generative information** input which helps designers to make decisions during the design process. Examples of information in this category include exemplars, rules of thumb, trade literature and so on.
- **Evaluative information** input which helps designers to evaluate, revise and refine their design decisions. The main examples of this type of information input are calculation algorithms and standards.

However, this division is not absolute. One information source could be used by one designer as a support to create ideas while to evaluate ideas by others. The information designers like to have at the early design stage includes exemplars, rules of thumb and algorithms (Kealy, 1988; Marvin, 1985b). Table 3.3 shows the likely purpose of these information sources.

Table 3.3 Usage of different information

	Exemplars	Rules	Algorithms
Generative	✓	✓	
Evaluative		✓	✓

Exemplars: Exemplars of graphic images such as pictures, videos and films, together with informative comments could enable designers to learn design expertise from others' experiences. Marvin (1985b) found designers' common preference for graphic images over text.

Exemplars could include whole building design, such as the introduction of exemplar designs in the Architects' Journal, or building element specification. They provide references to designers who have similar tasks in hands.

The potential of using exemplars as means to promote good practice have been realised by government organisations and research institutes. Some projects have been initiated, which were aimed at producing some good show case buildings, for instance the ETSU passive solar house design project in the UK (Riddle, 1988b).

Rules: A set of rules for building design, based on current knowledge, can provide concise advice at the early design stage. The rules include legislation, general design advice and specific strategies for specific design issues, i.e., site planning, building arrangement, building fabric and components as well as building services. Legislation can be considered as compulsory rules which must be met. For instance, one rule may read "the U-value of the external wall of a house cannot exceed 0.45 w/m²k". Since this is a requirement of the Building Regulation for England and Wales, it has to be complied with. General design advice and specific strategies are suggestions which are considered to be good solutions under certain circumstances. For example, one piece of design advice on passive solar house design is "to minimise the north facing window area" (BRE, 1988).

Design rules should be written in concise and brief language. They could be very effective in supporting decision making at general level. However, it is difficult to cover specific design variables in the provision of rules, e.g., how to decide the actual value of the minimum acceptable north facing window area.

Algorithms: Algorithms consist of a set of calculation procedures. They are largely evaluative tools. The purpose of algorithms is to provide designers with tools to evaluate their design decisions. For instance, when designers make decision on window area, several implications should be considered such as daylight and heat loss. The CIBSE daylight factor calculation and BREDEM could be used for checking window size. They could be presented in various forms, i.e., tables, figures, manual calculation procedures or computer programmes. The CIBSE window design guide provides a set of graphs to enable the calculation daylight factor in a room. The BREDEM method is available in both printed tables for manual calculation and computer programs. (Anderson, 1988; EAS, 1989; NCL, 1989)

Concluding remarks: These information sources have different purposes in supporting designers.

- Exemplars show experiences other designers had when designers faced the same or similar problems.
- Rule of thumbs provides general guidance for the design decision making at an early stage.
- Algorithms provide designers with tools to evaluate their decisions. Using these tools, designers can check their solutions against design criteria.

3.5.3 Criteria for information provision

Some of the desired features for technical publications have been revealed in relevant studies, i.e., Marvin (1985b), Yoon (1987) and Brewer (1988b). They can be summarised

as valid and authoritative, interactive and responsive, accessible, and design directed. The requirements of each of these aspects are discussed as below.

Valid and authoritative: It is impossible for designers to check the validity of the technical information, and the information supplier must take on this responsibility. For example, the simulation theory of algorithms should reflect the real design situations. Similarly, design guides should rule out any ambiguities, the conditions for application should be clearly defined. Advice at the early stage should avoid being contradictory to that at the detail design stage, and vice versa.

Interactive and responsive: A design support tool should be responsive to designers' requests and communicate with designers in an interactive fashion. This feature is particularly important in the case of evaluative tools. When designers use evaluative tools, they supply input and expect an answer. Very often they want to make changes in some of the input parameters and see the impact on the evaluation result. The evaluative tools should be able to respond to designers' instructions with a minimum of time delay; otherwise, their usefulness and effectiveness will be reduced. One implication of this conclusion is that several tools for different design aspects should be operate interactively to reduce the time lag of shifting between different tools. The information exchange between tools should be maintained to the maximum extent.

Accessible: An accessible information source should be concise and well organised. Designers should not be expected to filter out useful information from a mass of unstructured materials. For computer based applications accessibility, to a large extent, depends on their user interface. A 'user friendly' interface should enable the user to feel able to operate the system without too many difficulties.

Design directed: Design advice should be design directed. For example, when design guides are given, practical solutions should be used to illustrate the principles rather than to present details of experimental procedures.

3.6 Summary

Based on the analysis in this chapter, the following conclusions are made:

- (1) Architectural design in general requires both creativity and rationality. Unlike art, architectural design has more external constraints. On the other hand, unlike science, the solution to an architectural problem is not unique, and very often cannot be found by purely rational analysis.
- (2) In design, the process of searching for a solution is endless, iterative and cyclic. The final solution is always a compromise under the given circumstances.

- (3) In current design practice, the majority of designers find it difficult to assimilate technical information, especially at the early design stage.
- (4) In order to improve designers' work at the early design stage, a combination of different types of information should be provided for both generative and evaluative purposes.
- (5) Powell (1988) identified four types of architects in terms of their learning styles. When preparing information for them, their different requirements should be recognised and taken into account.
- (6) The provision of information should meet the requirements of being valid, authoritative, interactive and responsive, accessible as well as design oriented.

CHAPTER 4

REVIEW OF EXISTING DESIGN SUPPORT MEASURES

4.1 Introduction

Chapter 3 examined the architectural design process (section 3.3) and outlined the designers' information requirements at the early design stage (section 3.5). This chapter reviews the current provision of design support measures or design support systems against the designers' requirements identified in section 3.5. The following issues will be addressed in this chapter:

- components of a design support system,
- categories of existing design support systems,
- evaluations of the existing design support systems,
- suggestions for future improvements.

In the following discussion, 'designer' is used specifically referring to 'architect'. Therefore, these two words are used interchangeably hereafter. The same applies to design support measure and design support system.

4.2 Components of a Design Support System

A design support system has two essential components, the knowledge content and the communication media. The content is the knowledge concerning some aspect(s) of design, for instance, "the U-value of external wall for domestic building should not exceed 0.45 w/m²k". This statement represents an item of knowledge which designers need to know. In order to introduce the knowledge to designers, a communication media has to be employed. The media could be 'speech', 'written material', 'computer application' and so on. The commonly used technology transfer media and their characteristics are discussed in section 2.6.

Figure 4.1 illustrates the relationship between knowledge, communication media, design support system and designers.

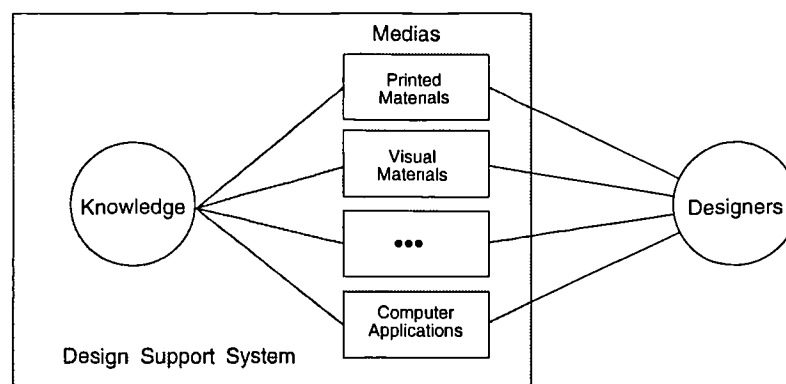


Figure 4.1 Components of a design support system

The knowledge or information content together with the communication media makes up a design support system; the success of knowledge transfer depends upon both having the appropriate knowledge content and using the most effective knowledge transfer media to maximise the information flow.

4.3 Classification of Design Support Systems

Design support systems can be classified according to either a content based approach or a media based approach. The content based approach classifies the design support systems according to the characteristics of the knowledge it contains, i.e., exemplars, design rules, calculation methods, etc. The media based approach classifies the design support systems according to the transfer media it uses, e.g., printed materials, visual materials, computer applications and so on. The difference between these two approaches is illustrated in figure 4.2.

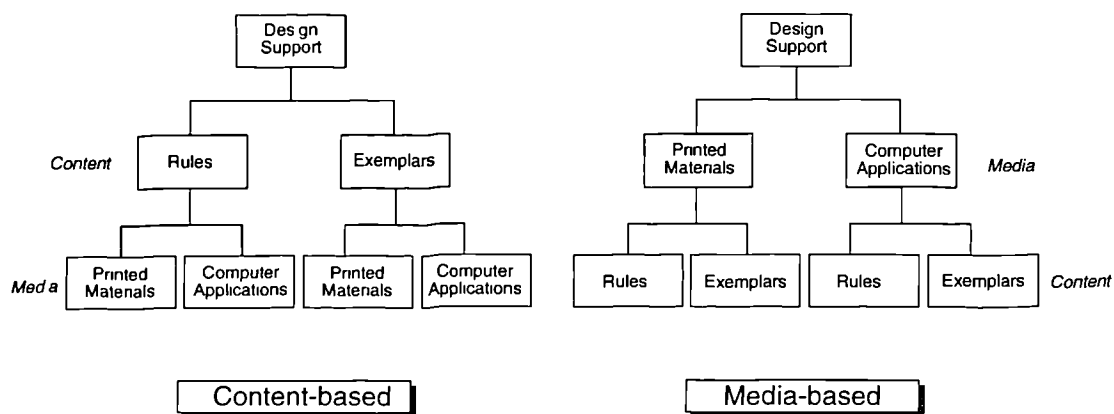


Figure 4.2 Two classification approaches

When the designers' information requirements were discussed in section 3.5.2, the emphasis was placed on knowledge content. The question then was "what kind of information do architects need?" Exemplars, rules and algorithms have been identified as being among the essential forms of knowledge required at the early design stage.

This chapter examines the current provision of design support systems. The question now is "in what form are the design support systems being presented to architects and how appropriate are they?" Therefore, the media-based classification approach is used.

The knowledge transfer media identified in section 2.6, except those requiring personal face to face contacts, can be broadly classified into three types, printed materials, visual materials and computer applications (figure 4.3). Each type of media can be further divided into specific application forms. It has to be pointed out that figure 4.3 is not an

exclusive list of all possible design support systems, those included are commonly used ones.

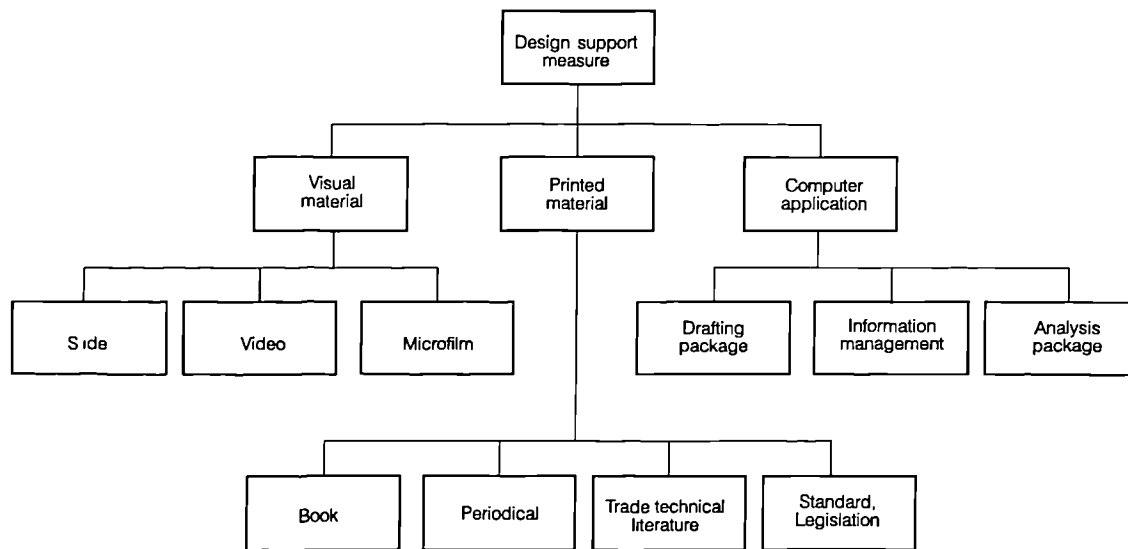


Figure 4.3 Classification of commonly used design support systems

In the following the characteristics of the different design support systems are discussed.

4.3.1 Printed materials

At the present time, printed materials are still the main information source for architects (Kealy 1988, Brewer 1988b, Marvin 1985a). These materials include books, periodicals, research reports, trade literature, government regulations and so on. The major advantages of this type of design support materials are: (1) they can be circulated to a wide audience; (2) most people are familiar with these communication media; (3) the content of printed materials can include both text and graphics.

"A wealth of written design and product information is published [in printed materials] or otherwise made available to designers" (Marvin, 1985a). There are about 2400 British Standard Codes of practice, several hundred architecture related journals and an enormous number of textbooks, research reports and technical publications of relevant subjects currently available to designers in the UK.

Books: Books are the most traditional means of knowledge dissemination. Books aimed at designers include textbooks such as *Design in Architecture* (Broadbent, 1988), reference books such as *Information Sources in Architecture* (Bradfield, 1983), handbooks such as *European Passive Solar Handbook* (CEC, 1988), and so on. A survey by Goodey (1971) suggested that a large proportion of design practices, more than 95%, have libraries

collecting relevant publications including books. The advantage of books as communication media is that they can be widely circulated and there are well-established retrieval systems for them, the library system. However, in practice, architects often find books less useful than they should be. There are various reasons for this; the most common complaint is that the knowledge in books is too general to be applied directly in design. Furthermore, a book is a relatively formal form of dissemination. It requires a long time to produce; the publishing process typically takes a few years and it is difficult to update due to the slow revision process.

Periodicals: One means of presenting the up-to-date information is through periodicals. The current monthly index book, Architectural Periodical Index (API) published by the RIBA, includes some 450 architecture related periodicals world-wide. In UK, the weekly periodical the "*Architect's Journal*" is probably the principal news and technical medium for architects. It encompasses most aspects of the profession, spreading its coverage over current events and future activities while providing longer feature articles and studies, including descriptions of new buildings, technical construction matters and professional practice articles" (Turner, 1983). *Building* is also a weekly periodical which is aimed at the construction profession and design team. Some other publications also quite widely read among architectural professionals, for example, *Building Design*, *Architectural Design*, *Architectural Review* and *RIBA Journal*. There are some other journals which are more technically specific, such as *Building and Environment*, *Concrete Quarterly*, *Light and Lighting Environmental Design*. Research conducted at the IOAAS of the York University (Marvin, 1985a) confirmed that most designers spend some time on reading related journals. However, in most cases it is only for general information rather than searching for solutions to a specific problem.

Trade literature and technical publications: Trade literature is publication about specific products, normally supplied by manufacturers. The information content in these publications usually includes not only a detailed description of the products and conditions of their uses, but also compliance with relevant standards and information on testing procedures. Surveys by BRE (Brewer, 1988b) revealed that trade literature formed the major bulk of references consulted by designers at the detail design stage. In some cases it is the only source of information consulted in addition to the designer's own experience. Designers use trade literature for choosing products as well as for learning about the general principles of the use of that type of product (Marvin, 1985a).

Brewer (1988b) found that the majority of designers considered trade literature to be "technical" information. However, there is a difference between trade literature and technical information. The former is aimed at promoting particular products though sometimes accompanied by some technical background. The later is aimed at the

dissemination of technical knowledge in order for it to be applied ultimately by designers in practice. Trade literature is generally used at the detail design stage when considering the use of particular products, while technical information can help designers when they form and evaluate their design concepts at the early design stages. Distinguished examples of technical information are the publications of Building Research Establishment (BRE), such as *BRE Information Paper*, *Digests*, *Current Papers* and *Reports*.

The size of technical publications may vary from a few pages, i.e., BRE information paper, to several hundreds of pages, such as *Information and experience in architectural design* (Marvin, 1985a). The issues addressed in technical publications could include applications of building technology, studies on ways in which building design and construction are conducted, investigations into building failures and so on. Though technical information such as BRE publications have a very wide circulation among design practitioners, its utilisation is not always satisfactory.

Government publications, standards and legislation: The most important official publications concerning the building industry in the UK are the British Standards and the Building Regulations. British Standards are advisory publications, which recommend minimum standards of good practice. They do not have the force of law but are a good defence in liability cases.

The Building Regulations are aimed at implementing the Parliament Acts related to building with the ultimate objective of ensuring the health and safety of the building inhabitants as well as the conservation of fuel and power (Elder, 1989). Its requirement is mandatory and building designs have to comply with it. In the approved documents of the Building Regulations, underpinning the requirements set out, references are made to other sources such as the British Standards, CIBSE Guide, etc. For example, in the 1990 edition of the approved document of the Building Regulations, for the calculation of building annual energy consumption the following reference is made, "in the case of buildings other than dwellings, in accordance with the space and water heating requirement section of CIBSE Energy Code 1981, Part 2a (worksheets 1a-1e)" (DoE, 1990). Given its importance and the compulsory nature of its requirement, Building Regulations, together with British Standards, are the most frequently used information sources by designers in practice (figure 4.4).

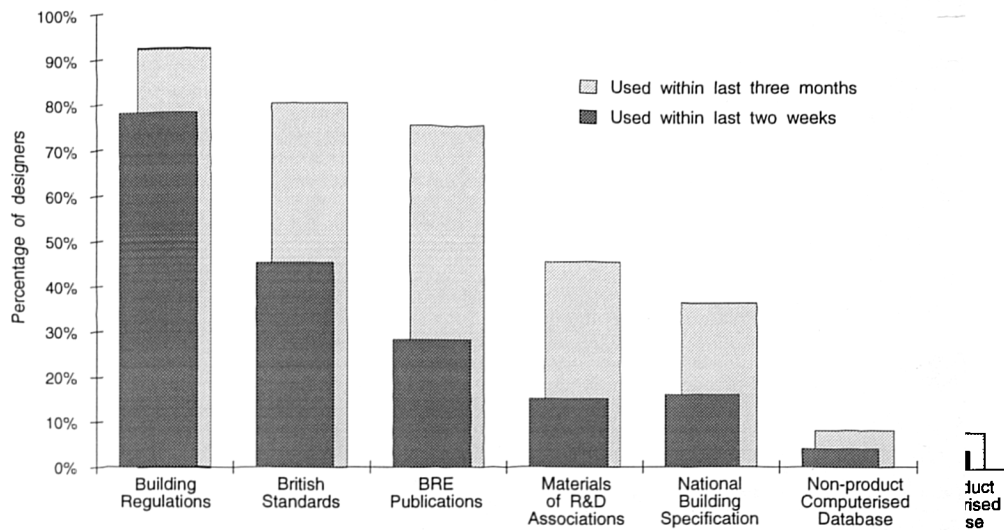


Figure 4.4 Frequency of use of various technical publications (after Brewer, 1988b)

Figure 4.4 shows that nearly 80% of the designers had used Building Regulations within two weeks of the survey date; and more than 90% of the designers had used them within the last three months. British Standards and BRE Publications are sources of good practice and sometime referenced to the Approved Document of the Building Regulations. Both of them are also frequently used by designers in practice.

4.3.2 Visual materials

Theoretically, all the design support materials used by designers, graphics or text, are visual. However, in this discussion, the term 'visual material' is used exclusively for those which require special visual equipment, i.e., slide project, video player. Many visual materials present information in form of moving or static graphics rather than text. Numerous studies (Goodey 1971, Mackinder 1982, Marvin 1985a, Brewer 1988b) have shown the architects' preference of visual images over text. Although visual images can be presented in printed materials, several other types of visual equipment can be used to present visual images and to achieve effects which cannot be implemented using printed materials. Commonly used visual materials include microfilms, microfiches, photographs, slides, video tapes, and films.

Microfilms and microfiches are used to store a large amount of information content in a small physical space. They facilitate the task of large quantity information storage and retrieval. Product Index and Technical Index provided by Barbour Index Ltd are good examples of the use of microfiches.

Photographs and slides are very good at displaying completed buildings and the surrounding environments. Most architects have private collections of photographs and slides of their own and others previous work. These materials are often used as exemplars

when dealing with new design projects. Some commercial slides on various topics are also available from some publishers, such as The Architecture of Urban Landscape supplied by Pigeon Audio-visual branch of World Microfilms (Bradfield, 1983).

Video tapes and film materials display moving pictures and introduce the time dimension to the viewer. They have been frequently used in education institutes. However, there is no evidence to suggest they are as widely used in design practice.

Generally speaking, visual materials are good at demonstrating technical principles by using positive and negative exemplars, with the exception of micro material which is effectively printed material in a micro form. Visual images presented by slides and videos are useful when designers formulate design concepts at the early design stages. However, it is difficult to provide dynamic help to individual projects. This is due to the lack of interaction between the user and the existing visual materials.

In recent years some research efforts have been made to try using video facilities as an effective means to present technical information to designers. Lewis (1988) introduced a study on an interactive video information system for energy efficient building design. The study explores the potential of interactive video technology for storing, retrieving and presenting information relevant to passive solar building design. Video facilities and computer system are linked to present information to the user in a 'friendly' manner. The positive results of this kind of effort have yet to be judged by practical application.

4.3.3 Computer applications

Another increasing important type of design support system is through the use of computers. The application of computing technology in architectural practice is by far the most important development in the last few decades. In the 1970s, many people were so impressed by the potential impact of computers that optimistic predictions were made, for example,

"...the theory and practice have developed to the point where it can confidently be predicted that, during the 1980's, every-day use of CAD techniques will radically transform the practice of architecture" (Mitchell, 1977).

In fact, this kind of optimism has not materialised for various reasons and the full potential of computing technology has not yet been fully realised.

At present most computer applications in architecture are aimed at very specific domain of design such as drafting, job management and discrete lighting or energy calculation. The deficiencies of the current packages are widely acknowledged and will be discussed in

section 4.4 (Brambley 1988, Ohsuga 1989, Mcdermott 1982, Sanders 1989).

Nevertheless, the impact of computers on architectural practice is far-reaching and computer applications in architecture have shown an increasing trend in recent years. A survey in 1989 by the RIBA suggested that most architectural design offices possess computers (table 4.1). This survey was the third in a series; the previous two being conducted in 1980 and 1987. Significant increases were observed both in the percentage of computer ownership and in the use of software applications. The most encouraging sign was that the majority of architects identified computer usage in their offices as a key area for the future development of the profession (Ray-Jones, 1990). Surveys in the United States have revealed a similar result (Brambley 1988, Stevens 1991).

Table 4.1 Computer usage in practice (% of all practices) (after Ray-Jones, 1990)

Number of architectural staff	1-2	3-5	6-10	11+
Have Computers	48	66	79	96
Use computers for				
- word-processing	44	61	70	90
- specification	25	37	41	56
- general accounts	18	23	31	57
- 2D CAD	11	20	32	62
- 3D CAD	11	18	28	53
- office costing	9	22	34	59

Although the computer has not yet radically transformed architectural practice, they have brought new dimensions to it. Architects no longer have to rely on paper drawings and experiences to predict the final appearance and performances of the building. Computer simulation software packages are now being used to display the building and to evaluate it against various performance criteria.

Initially, one of the major computer applications in architectural design is for drafting and many CAD systems for architectural design have been developed since the 1960s. (Sanders, 1989) Most of these systems were limited to the description of design, and the production of design drawings. This has not been proved to be a very effective strategy (Kalay, 1985). The result in practice is normally that senior architects still do the design work in a traditional way, then hand over the paper drawings to CAD operators, junior architects or draftsmen. The latter transform the paper drawings into computer ones through a very time-consuming process. This kind of CAD technology is often described as using high cost facilities to do low cost work, and not particularly effectively in many cases (Stoker, 1991).

The real advantage of a computer is its speed in manipulating and searching large amounts of data not merely in reporting it. This has been acknowledged by the developers of design support tools. A variety of analysis packages have begun to gradually emerge, which are used to predict building performance in lighting, energy consumption, acoustics, etc. For example, as early as October 1985, Architectural Record produced a Guide to computer software for architects and engineers which collected some 360 entries of computer software for various aspects of architectural design. The real scale of the development of computer applications for architectural design goes far beyond this list.

These discrete packages, even relatively sophisticated ones, are normally used for evaluating building design apart from the normal design process. The lack of integration with the design process becomes an increasingly obvious obstacle for their application in practice. These limitations are discussed in section 4.4 and section 5.3 of Chapter 5.

Since the 1980s, awareness of the need for integrated building design systems has been increasing with a number of research projects underway world-wide, i.e. in Finland (Björk, 1987), in the United States (Brambley, 1988) and in the European Community (Augenbroe, 1988). These research efforts focus on the integration of different tasks into one design support system in order to reflect the nature of the architectural design process more closely, and subsequently facilitate their usage in practice. Developments in this area will be the main theme of discussion in Chapter 5.

4.3.4 Summary of current design support provision

Table 4.2 presents a summary of the current provision of the design support systems to designers in practice. It is arranged in a matrix of communication media and information content. The design support systems mentioned in the table are not exclusive but representative examples.

Table 4.2 Summary of current design support provision

	Printed materials	Visual materials	Computer applications
Exemplars	Example projects introduced in related journals, such as Architects' Journal, Building and so on. Manufacturer supplied publications on specific materials.	Slides, films and videos of exemplar buildings. Microfilm or microfiches on product information.	CD-ROM of exemplar buildings. CD-ROM of manufacture products.
Rules	Regulations, Standards, Handbooks and so on. BRE publications.		Computer based information systems.
Algorithms	Regulations, Handbooks, BRE publications, etc.		Evaluative tools such as ESP, BREDEM.

Printed materials still make up the majority of the design support systems aimed at architects in practice. Professional journals such as *Architects' Journal*, *Building*, are the major information sources that architects rely on to follow the architectural and technological development in the building industry. In the U.K. the Approved Documents of the Building Regulations, British Standards and publications of the Building Research Establishment are widely circulated in the design professions. Designers trust these information sources as authoritative and valid. When deciding the use of specific products in design, manufacturer supplied trade literature is often consulted by designers. In addition, there are many design handbooks on specific subjects such as passive solar technology available to designers.

Microfilms and microfiches are used as alternatives to printed materials. Since large amounts of information can be stored on these materials, they are very useful to present information such as product databases. However, they are not able to present information any different to printed materials. Videos are commonly used to demonstrate exemplar buildings or for product promotion.

At present, the most widely used computer applications in architectural practice are specification writing and CAD drawing. These systems have limited value in terms of technology transfer since they do not include the transmission of technical information to designers. Evaluative tools, i.e., the ESP simulation program, involve underlying knowledge transfer. However, there are existing problems hampering their wide application, which are discussed in the following.

4.4 Evaluation of Existing Design Support Systems

The essence of a design support system is to provide information to designers as they need it during the design decision making process. To achieve this end, the necessary requirements for design support systems, as discussed in section 3.5.3, are that:

- They must be valid and authoritative.
- They must be interactive and responsive to designers' requests.
- They must be accessible and easy to use.
- They must provide help in making design decisions.

Although many design support systems are available at present, most designers in practice find it difficult to make full use of them, because they do not meet the above requirements. This under utilisation problem has been acknowledged for some time and series of studies have been conducted, which aimed at understanding the problems and improving the situation (Goodey, 1971; Mackinder, 1982; Marvin, 1985a,b; Yoon, 1987; Brewer, 1988b; Kealy, 1988). Two interrelated aspects have been highlighted in these studies.

- (1) the limitations of design professionals in utilising up to date technology.
- (2) the deficiencies of the existing design support systems in meeting designers' information requirements.

Chapter 3 has addressed the first aspect of the problem. In the following section, the focus is on understanding the deficiencies of current design support systems in order to identify the areas which must be addressed in the design of future design support systems.

4.4.1 Validity and authority

Designers in practice have a tendency to rely on authoritative publications in conducting their work. One proof of this is that the Building Regulations and British Standards are found to be the most widely used technical information sources (Brewer, 1988b). Designers consider the information in these publications as valid and reliable. Brown (1988) also suggested the lack of application of new technology, such as passive solar design and other energy saving techniques, in design practice is at least partially due to the lack of clear and authoritative design guidance. Without such authoritative guide, designers have to bear the liability risk of potential failures, and designers are keen to avoid this.

Validity is an unsolved issue for many existing design guides and computer software since the validity of the knowledge base is often undecided. For example, a handbook for passive solar house design was developed in 1988 (BRE, 1988). The advice in this handbook was based on simulated performance results of house designs using the thermal simulation model SERI-RES and on monitored results of exemplar building of low energy design. However, this basis is open to doubt as argued by Bloomfield (1988)

"in practice there are uncertainties associated with both predicted and measured values - those due to measurement inaccuracy of the output variables and those due to the effects of uncertainties and errors in the assumed model input parameters. ... (And such) an empirical validation study can only ever give information on model performance for one highly specific set of conditions - building type, operating conditions, climate, etc."

Consequently, design support systems developed using such an approach might be valid in some circumstances while not in others.

Invalidity is also caused by the difference between the conditions under which a design support system is developed and those under which it is applied. For example, Brewer (1986) reviewed the design handbooks on passive solar design available to UK designers. He concluded that most of the widely circulated guidance on passive solar design was

produced for North America conditions which did not always apply to the UK.

4.4.2 Interaction and responsiveness

The relationship between designers and a design support system or a design tool is that designers raise a request to the system and the system produces an answer (figure 4.5). The requirement on interaction and responsiveness means design support systems should support a convenient communication with the designers.

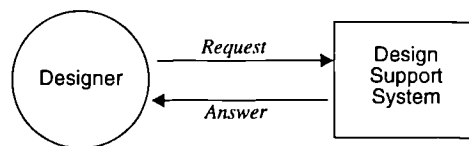


Figure 4.5 Interaction between designers and tools

This requirement is more important in the case of evaluative design support tools than generative tools (section 3.5.2). In the use of an evaluative tool, the tool needs input from the designer and the designer has to understand the evaluation results and the implications in term of design solutions. Some existing computer applications are very difficult to use because they do not have good user interface which conforms to this requirement. They ask for input and present output in ways that designers do not understand. When using them, designers often do not know what to do or what to make out of the evaluation results (Warren, 1992).

It is difficult to get the interaction and responsiveness in the case of printed materials, since the information is presented in a static matter. They cannot respond differently to individual designers' requests, and this is a major limitation of printed materials as a design support tool.

Many existing computer based design support tools address a single aspect of building design. In contrast, designers have to consider multiple aspects interactively during their decision making process (Lawson, 1980). Because each tool has its own perception of the design problem, there are conceptual barriers between different tools (figure 4.6). The discrete nature of design support provision increases the difficulty for the designer in understanding and utilising the tools provided.

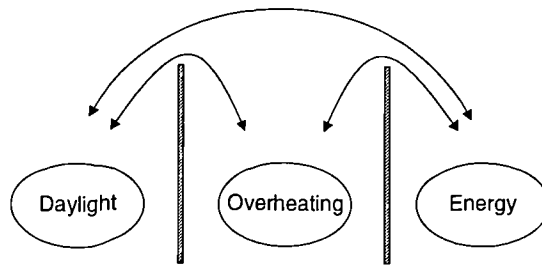


Figure 4.6 Discrete provision of design support tools

Under these circumstances, at best, the designer's interaction with the design support tools is constantly interrupted by the need to shift from one information source to another, which would result in inefficiency. At worst, the designer may simply give up the effort of using these tools, as a result, the technology transfer process fails.

4.4.3 Availability and accessibility

Although there is a massive knowledge base potentially available, architects often feel that obtaining access to the required knowledge content is too time-consuming and costly (Yoon, 1987). Many architectural practices, especially the smaller ones, do not always have sophisticated information retrieval systems. The technical publications available are often circulated randomly amongst staff. Architects are generally unwilling to spend time looking for technical information due to work load and time pressure. As a result, in practice some of the design support information intended for designers may be physically unavailable to individual designers.

Apart from the lack of availability, accessibility to the knowledge content is often another problem. In many cases, where designers find the publication which appears to be related to their problem, they do not have the knowledge to understand and use the information content provided. Designers often find it hard to draw practical solutions to their problems from technical documents. Complaints about the technical information being too scientific are quite common amongst architects (Marvin, 1985a).

The lack of accessibility of many technical publications is partially caused by their inappropriate presentation format. Related studies have shown that designers prefer precise, specific materials, especially with illustrative graphics. Unfortunately, many of the existing technical publications do not have these features. "Information is often unavailable in a concise and readily understandable form suitable for using directly into problem solving, i.e., it is lengthy, wordy or circumlocutory" (Yoon, 1987).

The requirement for conciseness does not mean every technical publication should not

exceed four pages. In fact, the length of the publications was considered less important by many designers, this was confirmed by a survey (Brewer, 1988a). "Not more than four pages" came last in the thirteen most desired features of technical publication. What mattered is that the information should be arranged with a clear and concise layout so that designers can go through it easily and find out what they want.

The problem of accessibility is more acute for many computer based design support systems, such as ESP, or SERI-RES. Ordinary architects often lack the essential training and knowledge to operate them. As a result, the software is often used by specialists separated from the design process. This seriously decreases the effectiveness of their application in the design process.

4.4.4 Design oriented solutions

Designers often work under time pressure in practice. What they need from a design support system are solutions to their problems. Many existing design support systems fail to meet this demand (Augenbroe, 1989b).

One cause of this problem is the diversity of information requirements by different audiences such as building researchers, educationists, design practitioners and the confusion of target audiences. Some information materials circulated to designers are not specifically to their needs but more suitable for researchers or other professions. Therefore, it is not surprising that designers often complained that some technical publications are too scientific and lack practical design solutions (Marvin 1985a, b; Mackinder 1982; Brewer 1988b). It is obviously impossible for each and every technical material to provide specific design solutions to individual design project. However, what needs to be done in this respect is a brief and clear introduction to the conditions within which the information applies.

Design advice provided by some current printed materials is also very difficult to apply in design due to ambiguities. For example in a design handbook (BRE, 1988) for passive solar house design, one strategy offered to reduce building energy consumption is "to reduce north-facing glazing area and increase south-facing glazing area, and it is more important to reduce the area of north-facing glazing than to increase the area facing south". However, this statement can only be true under certain circumstances, such as using a specific glazing type and the glazing area is within certain range. Further more, by blindly adopting this strategy, designers may risk causing overheating in south facing rooms or inadequate daylight in north facing rooms.

Design support systems for single aspect of design, i.e., lighting or thermal, are also often

difficult to apply in practice. Architectural design is an interactive process which "requires the embracing of all aspects of the problem immediately - where gaps in knowledge exist they must be solved in an ad hoc manner with insight and experience. A designer cannot wait to have these gaps filled by an analytical effort, at least during the design process itself" (Bryan, 1986). Bryan identified the difficulties of applying discrete tools which support single aspect of architectural design. Designers do not like to evaluate a single aspect of the design problem without knowing the implications for the other aspects.

4.4.5 Concluding remarks

Based on the above discussion, the following conclusions can be drawn on the deficiencies of the existing design measures for architects in practice.

- There is a lack of validity and authority underpinning the knowledge base of many design support tools.
- It is difficult to maintain interactive communication between paper borne design support tools and designers.
- The single facet nature of many design support tools and the lack of underlying conceptual integration make it practically impossible for designers to consult multiple tools interactively during the design process.
- The provision of discrete design support tool results in scattered information sources which are not always available to individual designers.
- The lack of accessibility to some of the design tools, especially computer based applications, is caused by the difficulty in operating them.
- Another common deficiency of most existing design support tools is the lack of design oriented advice.

Improvements are needed in a number of areas in order to improve the effectiveness of the design support practice and subsequently the technology transfer process.

4.5 Recommended Improvements

Section 2.9 discussed the improvement of the technology transfer process at the general policy level. This section discusses specific measures which could be adopted to improve the effectiveness of design support measures in relation to:

- (1) making existing information from disparate sources available to designers?
- (2) helping designers integrate technical information into their design solutions?

In the near future, printed materials will remain as one of the major communication media for the technology transfer process in the building industry. Several steps can be taken to

improve its effectiveness.

- Improved presentation. A survey had shown some of the presentation features desired by designers, A4 size, date of issue, cost information, source of further information, summary of contents, informative title, many illustrations, CI/SfB coding and full-colour illustrations and so on (Brewer, 1988a).
- Concise content. Any information aimed at design support must be concise and easy to absorb. Designers should not be forced to filter out a solution from a mass of uncoordinated material. (Mavin 1985b, Kealy 1988)
- Design oriented. Practical implications of the underlining knowledge should be demonstrated using design solutions. Design support for important design issues, i.e. condensation, ventilation, insulation, could be documented in special publications. However, the information must be presented in a design context not in isolation.
- Wider distribution. Some technical materials, such as BRE publications, should ideally be distributed free of charge in order to maximise their circulation (Brewer, 1988b).

Although improvement is possible in the aspects of validity and authoritativeness, availability and accessibility for printed design support materials, it is difficult to achieve interactiveness and responsiveness to individual designer's need, due to their static nature. Evaluative tools cannot be presented effectively in printed forms. Similar constraints apply to the traditional visual material, such as slides, films and video. Research on interactive video information system, e.g., the one introduced by Lewis (1988), might bring some breakthroughs in the application of video facilities in the technology transfer process.

Despite the deficiencies of the existing computer based design support tools, computer applications appear to offer a promising solution to the technology transfer problem in the building industry. With the rapid improvement in the hardware and software, computer based design support tools will be able to compete with printed and visual materials, and they will have advantages in several important respects.

- Just as the paper borne media, computer is capable of presenting valid, authoritative and concise technical information to the point of use.
- Using computers, large mounts of information from various sources can be compiled together and presented to designers as a single integrated package.
- Two-way communication is possible between designers and the design support tools in the case of computer applications. This offers the potential to develop better interactive and responsive evaluative tools.
- The computer also provides the possibility of integrating multiple design support tools seamlessly together (figure 4.7), to break down the barriers between different tools as indicated in figure 4.6.

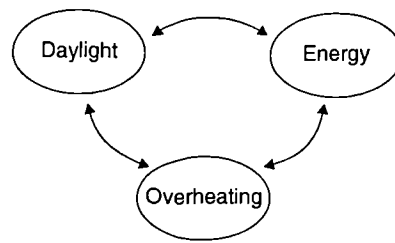


Figure 4.7 Integrated provision of design support tools

Making the potential benefit offered by the application of computers into a reality in the technology transfer is an important issue which has wide ranging implications. The more focusing issue of this study is how to improve the architectural design support with the help of using computers. Chapter 5 explores various aspects of this subject.

4.6 Summary

Many existing design support systems are not effective in providing design support for designers in practice. Some of the major problems have been discussed in this chapter.

The fundamental requirement of design support is to provide information which responds to individual designer's needs. This demands massive information storage and fast manipulation. Printed materials and existing visual facilities do not have the ability to meet this demand. The increasingly application of computer in architectural design practice (RIBA, 1990) offers a promising prospect.

CHAPTER 5

DEVELOPMENT OF COMPUTER APPLICATIONS IN ARCHITECTURE

5.1 Introduction

Chapter 4 concluded that computer applications have considerable potential for the improvement of architectural design support, and hence, technology transfer in the building industry. This chapter focuses on recent developments in this area. Major issues covered in this chapter are:

- the potential offered by computers,
- achievements and problems of existing computer applications,
- the development of integrated building design systems,
- some outstanding issues in the development of integrated building design systems.

5.2 Application of Computers

When people discuss the application of computers, 'revolution' is the word which is often used. Indeed, the impact of computers on the society in just a few decades can match that made by the industrial revolution in a few centuries (EDC, 1987). During the last 40 years, computers have been changing the way people live and think, and today computers are playing important roles in many areas of our lives, i.e., transportation, commerce, agriculture, education, science, the building industry, etc. Since the mid 1980s there has been a significant advance in computing technology which opens new opportunities for its wider application in architecture.

5.2.1 The computing environment

The understanding of most ordinary users of the computing environment is limited to input and output devices and a number of operational procedures. There is no reason for them to know more, why should using a computer be any different from driving a car? The computing environment is, of course, very complex as illustrated in figure 5.1.

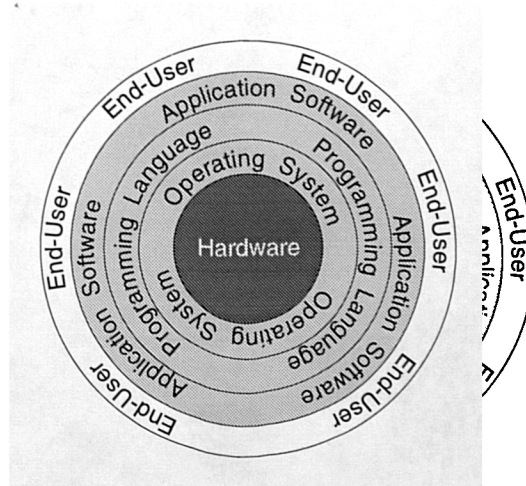


Figure 5.1 Components of the computing environment

The computer environment consists of hardware and software. Hardware refers to the physical devices for inputting, outputting, processing and storing data. Software refers to the rules or instructions which instruct the hardware how to behave. The software part can be further divided into operating system, programming language and application program. Each of these components has a direct impact on the performance of the computing system.

Hardware

The computer hardware is a device that accepts data (input) and processes it into useful information according to instructions (Capron, 1990). It includes four basic components each of which performs different functions, input, output, process and storage (figure 5.2).

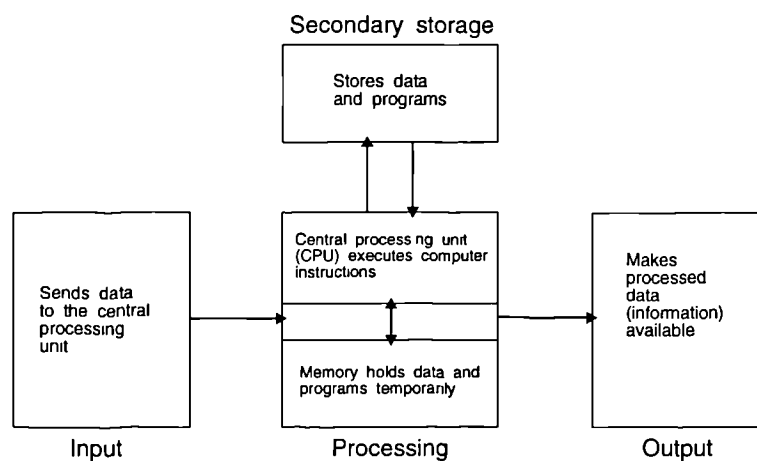


Figure 5.2 The structure of computer hardware (after Capron, 1990)

The performance of the computer hardware depends on the capacity and quality of all these four components, especially storage and processing.

Since the first commercial computer, the UNIVAC, was produced in 1951, computer hardware has experienced four generations (table 5.1), each of which was marked by significant performance improvements (Slotnick, 1989; Capron, 1990).

Table 5.1 Generations of computer hardware

Generation	Date	Hardware
First	1951-1958	Vacuum tubes
Second	1959-1964	Transistors
Third	1965-1971	Integrated circuits
Fourth	1971-Present	Large-scale integration/very large-scale integration

At present, the computer industry is undergoing a rapid development with improvements in overall performance, at the same time the cost, especially the cost performance ratio, has been coming down dramatically (figure 5.3).

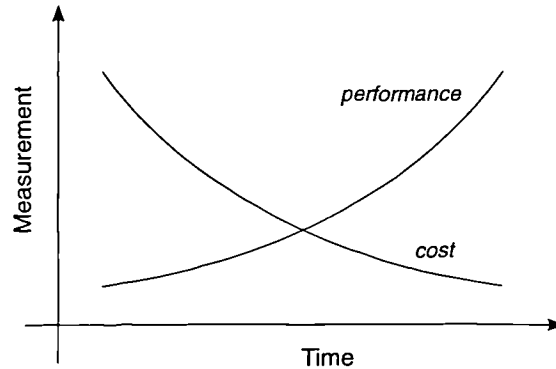


Figure 5.3 Change of computer cost and performance

Software

On the software side, there has been an equally remarkable progress during the past few decades. DOS and UNIX have emerged as the industry standard operating system for most of the hardware platforms. They act as media between programming languages and the computer hardware. There have been hundreds of programming languages; some have been dropped as more advanced ones are developed (Capron, 1990). Historically, programming languages have experienced four stages of development, machine language, assembly language, high-level language, and very high level language. With each advance, the programming languages are more capable of supporting complex logic; programming becomes easier and they allow greater attention to be paid to the conceptual issues involved. An increasing number of people with other domain knowledge have joined computing scientists, perhaps as a consequence more and more practically useful programs have been developed.

It is reasonable to predict that with the next generation of programming languages, natural language, the end users will be able to participant in the programming process and customising programs for their personal use.

Application software in architecture will be discussed in section 5.3.

5.2.2 What does the computer offer?

Computers offer the following benefits:

- **Speed.** A computer processes information incredibly fast, from several thousand to several billions of instructions per second. Without computers, some tasks, such as space exploration, missile precision targeting, could not possibly be carried out. Real time computer systems are able to respond to the instructing input with a minimum of time delay. The fast processing speed can help to increase the productivity of any task and to improve the man/machine interaction.
- **Massive storage.** During the past few decades, there has been a massive increase in information. At present, most of the tasks in scientific research, office work and building design involve handling large amount of information in various forms, i.e. text, numbers or graphics. The computer provides an excellent means of storing and retrieving large amounts of such information (figure 5.4).

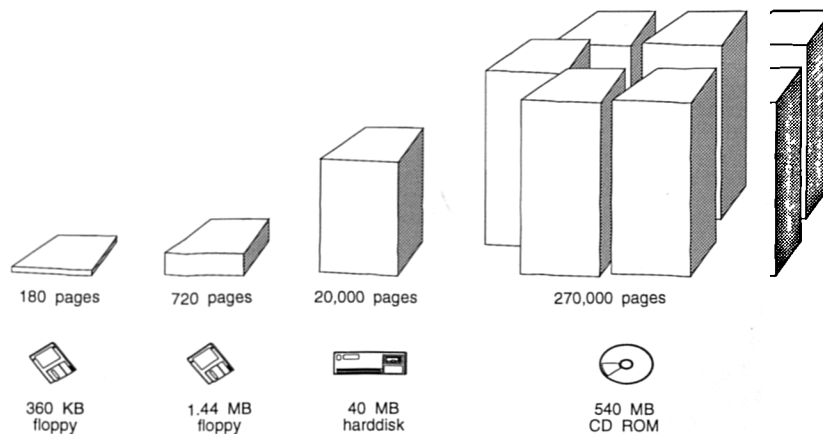


Figure 5.4 Storage capacity of several types of disk

- **Reliability.** Another merit of the computer is its reliability. The computer is extremely reliable at doing what it is told to do. It is true that sometimes computers have failures or crashes during the operation. However, these failures are usually caused not by the computer itself but by human errors during either the programming stage or in use.
- **Multiple Input and Output (I/O) devices.** There are a wide variety of I/O devices available such as key board, scanner, microphone, monitor, printer, plotter, etc., which can be used to read and present text, graphics and sound. The combination of the multiple I/O devices offers a rich capacity for developing application software.

5.2.3 Computers in architecture

For design professionals, the initial surge of enthusiasm for computer application started in the early 1960s. It was generated by an optimistic view of the computer's potential as a design support tool and the time needed to develop this potential (Stevens, 1991). By the 1970s and early 1980s, the initial excitement had been replaced by a greater realism about what the computer could offer. The change of opinion was caused by a combination of the high capital costs of computing hardware and the limitations of the existing design support software. In 1985, a AIA survey in the United States revealed that many architectural firms did not use computers because they believed "it would cost too much" (AR, 1985). On the performance aspect, there were the following constraints. (1) The small storage capacity limited the amount of information which could be stored. (2) The processing speed was relatively slow. (3) The I/O facilities were inadequate.

Nevertheless, the penetration of computer application in the architectural design profession continued to increase. Since the mid 1980s, the increase trend has been accelerating thanks to the rapid development in computer hardware and software. The cost of computing hardware is coming down while the performance, in term of memory size, processing speed, I/O facilities and so on, is improving rapidly as indicated by figure 5.3.

The rapid development of the Personal Computer (PC) and PC based local network have made a very significant impact on the application of computers in architectural design practice. At present, many PCs with several hundred megabytes memory are available at affordable prices, i.e., 2000-4000 pounds. Computers of this type can run most of the software needed by designers, such as information management systems, design analysis systems even architectural CAD packages.

As a result, there has been a steady increase of the computer ownership and application among architectural practices during the last decade. More and more design firms are willing to invest in computing facilities. Surveys in Britain show that the design practices having in-house computing facilities increased from 19% in 1980, to 47% in 1987 and 65% in 1989 (Ray-Jones, 1990). A similar trend in computer ownership by architectural design firms was reported in the United States, and it was predicted that the computer penetration would reach saturation point by the early 1990s (figure 5.5) (Stevens, 1991).

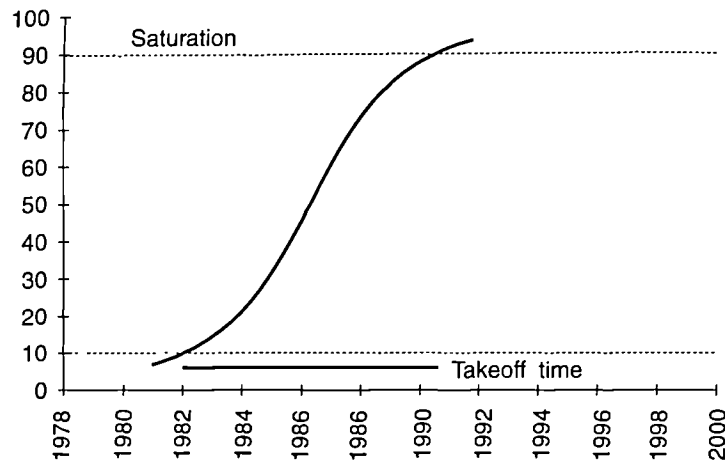


Figure 5.5 The penetration of computing into architecture firms (after Stevens, 1991)

The increasing importance of computer applications in architectural design is also reflected by the steady increase of research activities in this area. Professional periodicals, such as *Architectural Record* and *The Architects' Journal*, have papers on the development of new design support software almost in every issue. Figure 5.6 shows the increasing number of articles on computing in architecture included in the Architectural Periodical Index (API) as a percentage of the total literature in the respective year.



Figure 5.6 The proportion of articles on computing listed in the API (after Stevens, 1991)

Along with the increase in ownership of the computer hardware in design practices, more and more design support software packages are becoming available for various tasks, such as office management, project cost analysis and control, project scheduling and management, space planning and facility management, computer-aided design and drafting as well as building engineering.

5.3 Computer Applications in Architecture

The inspiration that the designer's ability could be enhanced in design with the aid of computing technology is a major driving force for the development of computer application in architecture. People are motivated by the prospect of combining the creative and imaginative power of the designer with the analytical and computational power of the computer to make the architectural design process more informative and productive (Coons, 1975). Bazjanac (1975) identified three areas where computer applications could offer major benefits:

- Computer drafting systems could free designers from distracting and unproductive activities allowing them to concentrate on the creative aspects of design.
- Computer analytical systems could support design decision making by enabling the designers to rapidly test and evaluate design alternatives in the search of the optimum solution.
- Computer information management systems can offer designers instant access to the accumulated knowledge in the building industry.

In the following, the achievements and deficiencies of existing computer applications in these three areas are discussed.

5.3.1 Computer aided design and drafting

Computer Aided Design (CAD), also known as Computer Aided Design and Drafting (CADD), has had the highest profile in computer application in building design. The majority of the literature on computing in architectural journals, such as *Architectural Record* and *The Architects' Journal*, is on this subject. In practice, more than half of the architectural firms with more than 11 employees in the UK use CAD packages (Ray-Jones, 1990).

The basic function of CAD packages is that they allow the user to build up drawings by manipulating lines, circles, rectangles and texts interactively on the screen. Some architecture specific CAD systems even provide graphical libraries of common used building elements, i.e., doors, windows and so on (Richens, 1990).

The initial drawing alone does not show the advantages of the CAD, since designers can draw on the drawing board equally fast. The real strength of CAD lies in its ability of allowing 'editing'. Once a graph is drawn, functions such as delete, move, copy, rotate, scale, mirror, etc., can be applied to any part of it. Other useful CAD functions are available such as, repetition of an element at equal distances along a line, around a circle or on a grid or matrix; extending lines, partial erasures of lines, insertion of fillets, etc.

(Richens, 1990). These tasks can not be easily carried out using paper based media without restarting from scratch. The easy 'editing' feature of CAD systems enables designers to explore more alternatives of building layout during design (Kharrufa, 1988). Furthermore, since the drawing can be saved at any stage, the designers are able to keep various versions of the building layout for later study. Once the geometrical information of the building design is stored in a CAD package, different views of the building can easily be reproduced (figure 5.7).

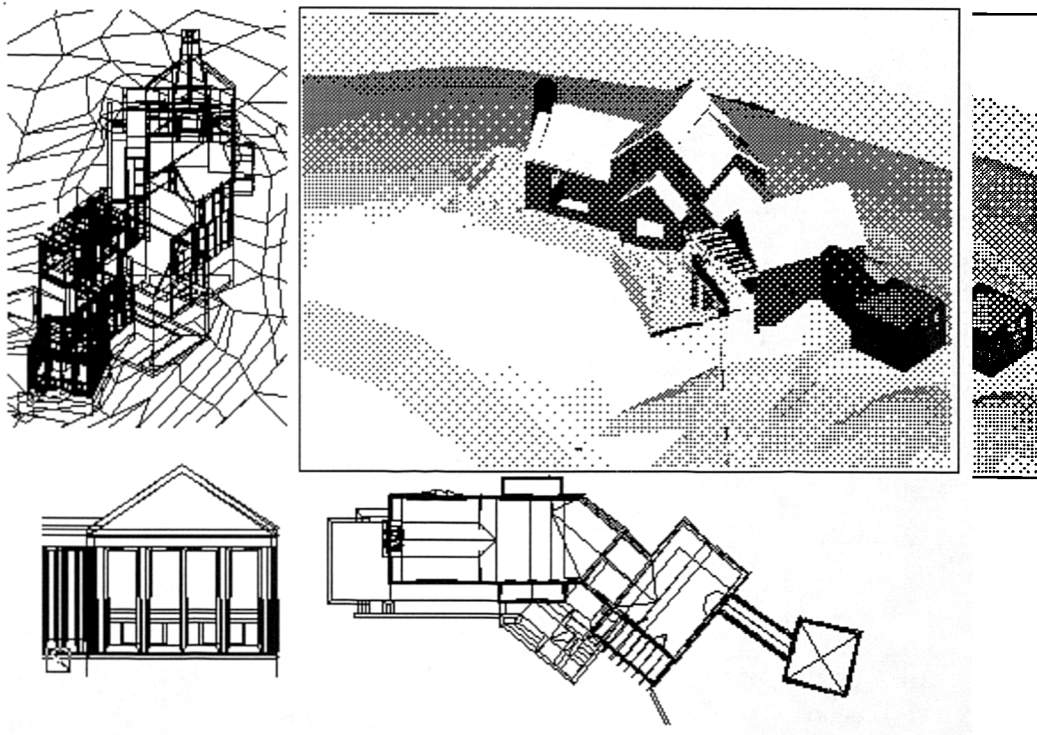


Figure 5.7 Multiple views of building produced by CAD

Despite the above benefits, there are some fundamental limitations in the existing CAD packages:

- (1) Most of these CAD packages are only suitable for drawing at the production stage when all the major design decisions have been made, not during the whole design process (Nicholson, 1990).
- (2) Current CAD drawing packages usually involve only geometrical aspect of the design (Oxman, 1991). The building is represented by points, lines, surfaces not by walls, windows and rooms. Construction information is not included. Therefore, it is difficult, if not impossible, to perform analytical evaluations on the building in the current CAD environment.
- (3) The user interface of many CAD drafting packages is complex. Designers often find it difficult to operate them effectively, as a result, they sometime can be distracted

from the 'design' task which they should really concentrate on. In some cases, specially trained staff have been employed to manage and use the drafting system. Such a practice increases the difficulty of integrating CAD applications into the design process. Although some of the latest release of CAD systems, i.e., AutoCAD release 12 for Windows, have improved in user interface, their operation is still time consuming.

While not undervaluing the benefits of computer aided drafting systems at the production stage, the above areas expose their weakness. CAD drafting systems are only drawing tools and they do not involve underlying technology transfer. They "will not make a great designer out of a poor one just as a word processor will not make an author." (Nicholson, 1987)

5.3.2 Evaluative design support tools

Evaluative computer applications are available for various architectural design aspects such as energy, lighting, acoustic, structure, HVAC design and so on. Discussion in section 3.3 shows that evaluation is one of the essential steps in the building design process. Unlike other engineering fields such as car and aircraft design where prototypes can be used for testing and evaluation before mass production, the prototype of the building design is usually the building itself. Therefore, it is all the more necessary for designers to be able to evaluate the building performance during the design process.

Although it is possible to carry out evaluations in a conventional and ad hoc manner without the aid of computers, it is difficult to take account of the numerous factors in the relationship between the building and the outside environment (Lansdown, 1984). Computer applications can be developed to perform dynamic simulations processing large amount of data and present the results with relatively short time delays (figure 5.8).

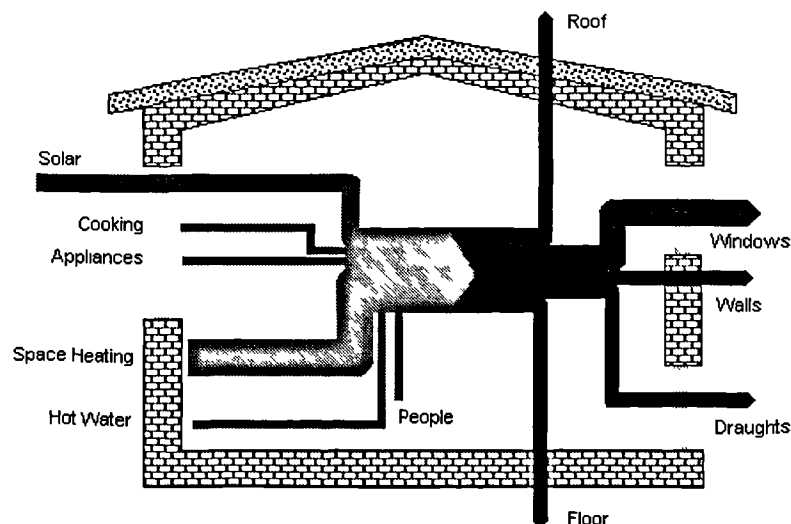


Figure 5.8 Example of computer evaluative program

The objective of evaluative computer programs is to enable designers to examine a wide range of possible design solutions and make qualitative comparisons between them (Lansdown, 1984). The most commonly used evaluative computer programs are for energy auditing and analysis (Stevens, 1991), structural design, lighting and HVAC are also used to a certain extent by designers in practice (Schmitt, 1988). However, Augenbroe, (1989a) pointed out that there are evidences showing that these existing programs are not fully used by the designers in practice. A number of underlying causes are suggested.

- (1) These packages are normally used for evaluating design solutions not providing design oriented advice. They require substantial detailed data input; therefore the operation is complex and time consuming, and considerable experience is required to translate the output produced by these programs into design solutions.
- (2) These programs are usually discrete packages each of which has its own format for input and output. There is no standard for the information exchange and data sharing between different packages. The integration within a system or between systems is very rare. The results of this situation are inefficiency in data management and the risk of data inconsistency.
- (3) The user-system interface tends not to be 'user friendly' and specific expertise is often required to operate the packages, which ordinary designers do not normally have.

Architectural design is an integrated process with many tasks need to be done interactively (section 3.3.4). Most of the existing evaluative computer applications are not able to support such interactive working. Although some individual software packages can perform single aspect of design support tasks very well, the operation of multiple applications demands extensive human involvement due to the lack of integration between different software packages. This seriously holds back their effectiveness and increases the risk of errors.

5.3.3 Information management systems

Information management is one of the major tasks in the architectural design process. Traditional paper borne information handling measures are not very effective. The massive storage capacity of computer hardware provides the possibility for developing better and more effective information management systems.

Two essential requirements for the information management applications are information storage and information retrieval. In architectural design, information from various sources is needed, for example, product information, technical information, regulation requirements, etc. An effective information system needs to compile and link this

information together in order to provide designers with easy access to otherwise scattered information. One example of this type application is *Specification Manager* (figure 5.9) (NBS, 1992).

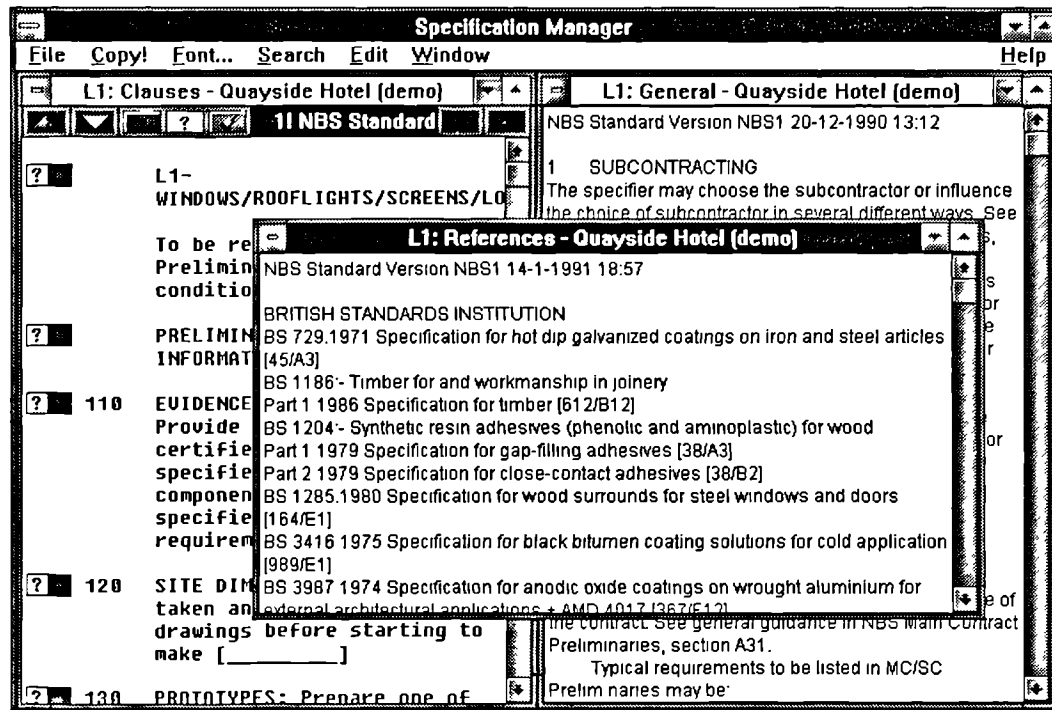


Figure 5.9 Example of information management system

Specification Manager is a comprehensive package performing several tasks in an MS-Windows environment. It has a main window to compile project specification, edit windows to edit clauses and card application to show manufacture and products information. These tasks can be conducted interactively and the information can easily be passed from one task to another.

Apart from providing access to pre-compiled information, some systems, i.e. Macintosh Hypercard, even allow the user to provide information input and customise the system.

Hypercard is a program and programming tool for connecting, organising and presenting all sorts of data (Leslie 1989). Its basic unit is a card or a frame of display on the computer screen. In a hypercard system, a set of card can be linked together to form a stack, and several stacks can be linked to form a larger stack. By doing so, an infinite collection of cards is inter-connected to form an information storage and retrieval system. The only limit is the memory size of the computer. On each card, information can be presented in text, graphics and even sound. The user can jump amongst cards by pointing and clicking with the mouse.

Since a hypercard system provides the potential for a user to create personalised information system without much programming knowledge. Lansdown (1990) has argued that a hypercard system is easy for ordinary designers to use, its application in architecture practice may increase in the next few years.

One common deficiency for the current available information management systems is, once again, the lack of integration. Most of them are not integrated with either CAD drawing packages or evaluative programs. The implication of this is that drawing, evaluation and specification have to be conducted separately, and the information exchange between these activities has to be done manually.

5.3.4 Other applications

Expert systems, also known as knowledge based systems or artificial intelligence, are another area of computer application in architecture. An expert system is a "reasoning system which attempts to mimic the performance of the reasoning expert." (Rosenman, 1987) Many expert systems have been developed for specific tasks such as element detail specification (Radford, 1985), compliance with practice code (Oey, 1988), roof design (Fazio, 1989), etc. These systems operate using rule-based reasoning methods to provide limited number of 'if-then' type of advice to designers.

The development of expert systems requires an effective knowledge acquisition process during which the knowledge has to be expressed explicitly. Unfortunately, in building design, designers are not always able to describe their knowledge with sufficient precision to a computer programmer (Fisher, 1986). Therefore, despite the initial enthusiasm, expert systems have failed to make a significant impact on architectural design practice (Ohsuga, 1989).

5.3.5 Conclusions

The above discussion reveals the current achievements and deficiencies of computer based design support tools in architecture. One of the common deficiencies is the lack of integration between different tools. At present, all the tools have their own perception of the building design. They require specific ways of data input and output as well as operating procedures. The use of one tool is often tedious and time consuming, using multiple tools during the design process is to multiply the difficulty.

Some computer application research projects being undertaken still have emphasis on particular aspects of design, for example, multiple visual presentations of the building (Fallon, 1990; Eastman, 1991), the information management (Evans, 1989; Leslie, 1989)

and the design performance analysis applications. However, there is a growing consensus in the research community that the development of computer application should adopt an integrated approach (Augenbroe, 1989a; Sanders, 1989). The objective is to develop Integrated Building Design Systems.

5.4 Integrated Building Design System

5.4.1 Overview

While research on integrated computer aided design started at as early as 1960s the development in this area was very slow for a long period due to the lack of adequate funding and the performance limitations of programming language and computer hardware (Vanier, 1987). Since the mid 1980s, the increase of computer applications in architectural design has exposed the inadequacy of the existing discrete application software. The development of Integrated Building Design System (IBDS) has consequently been placed high on the agenda.

The concept of an IBDS is that it will contain many tools for different tasks, which can run concurrently upon the same data model. These tools are conceptually integrated and interactive data exchange between tools is handled automatically by the system under the control of the user.

According to Sprague (1986), an integrated building design system has three essential components, a database, a toolbase and an interface management, in order to dynamically support multiple tasks (figure 5.10).

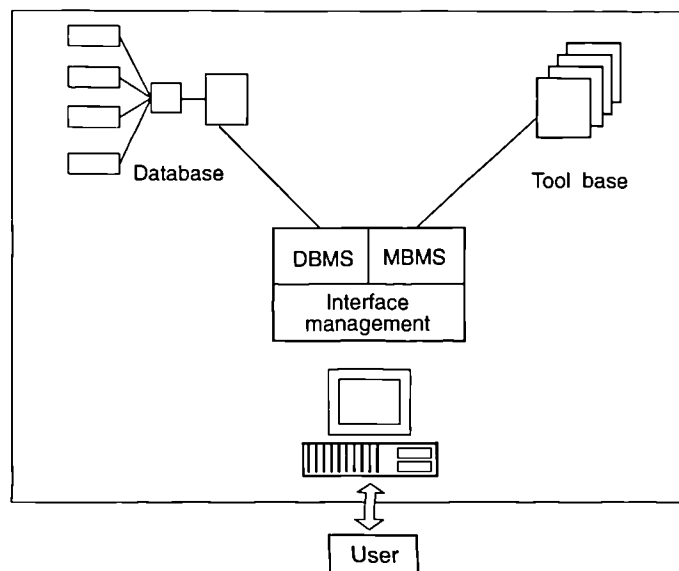


Figure 5.10 Structure of an integrated building design system

The function of each component in the system is as follows. The Database stores all the data or data slots to describe the object under investigation such as a building; the Tool base stores application tools related to the tasks that the system intends to support. Interface management handles the communication between the user and the system. The Database and Tool base are managed by Database Management Software (DBMS) and Model Base Management Software (MBMS) respectively. The combination of the DBMS and MBMS handles the selection of tools to use according to user's instruction and the data capture and extraction in order to operate these tools. Despite the similarities, there are variations in the system structure for individual IBDS systems, which are explained later in this section.

Augenbroe (1991) identified two different approaches toward the development of IBDS.

- Project-driven approach
- Object-driven approach

The differences and commonalities between these two approaches are illustrated in figure 5.11.

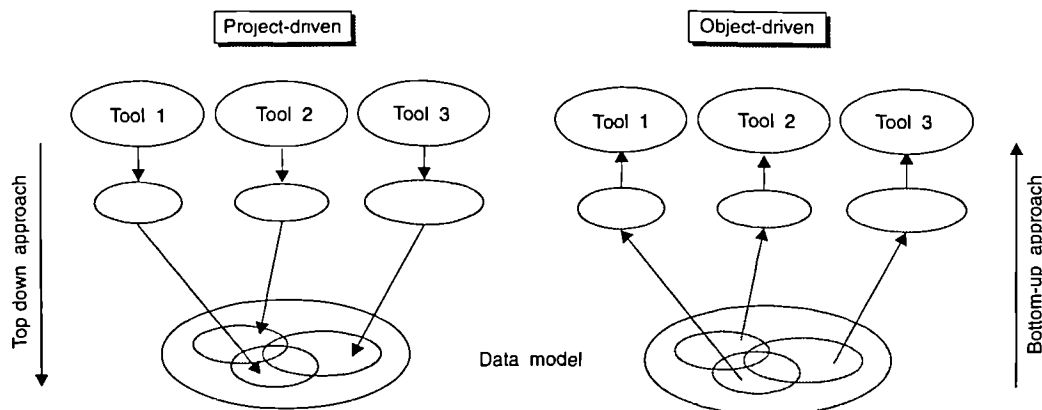


Figure 5.11 Two approaches to the development of IBDS

The project-driven approach is a top-down approach. Project in this context does not mean architectural design project but an IBDS development project. It is based on a more or less preconceived scenario for a limited class of design products, i.e., energy related issues for office buildings. The scenario pre-supposes a flow of design actions, each of which is assigned to specific components inside the system. The set of possible interactions is specified at the beginning of the project and the emphasis of this approach is on how to perform a limited number of tasks intelligently. The output of this approach is usually a closed system with a partial building data model which is only aimed at satisfying the tools included in the system (grey area in figure 5.11) (Augenbroe, 1991).

The object-driven approach, on the other hand, is a bottom-up approach. It emphasises the complete description of the building object in order to support all imaginable data exchanges among different tools. It describes the building as what it is rather than how it is used by individual tools. All the tools in the system map their data requirement on to the data model. The ultimate output of such an approach is an open system with a standardised building data model. Theoretically, an infinite number of tools can be added to such a system.

These two approaches are not conflicting but complementary to each other. The project-driven approach explores ways to produce practical and useful software applications. The object-driven approach provides a potential basis for future software development.

Since the late 1980s, a number of prototypes of integrated building design system have been emerging from related research projects, i.e., Advanced Energy Design and Operations Technologies of the USA (Brambley, 1988), Integrated Structural Design System of the University of Strathclyde UK (Macleod, 1988), Integrated Building Design Environment of the Carnegie Mellon University USA (Fenves, 1990), Computer Models for the Building Industry in Europe (Augenbroe 1989a, 1989b), etc. These systems can be classified into three categories (NCL, 1992).

- Database-centred
- Executive-centred
- Product model centred

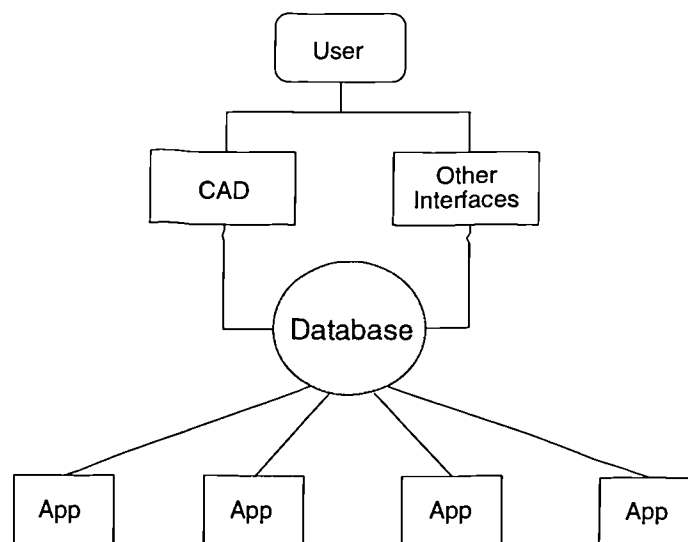


Figure 5.12 Database-Centred system

A **Database centred** system, as the name suggests, is a system with a collection of programmes or application tools located around a database (figure 5.12). All the

applications draw data from the database and put the results back to it. Therefore the database contains the following elements.

- a representation of the building covering views of all the tools included in the system.
- management modules for the communication between tools and the database.
- the results produced by the tools.

The database is usually implemented in a commercial database system and different tools can access it by using their own predefined query strategy. It is a fairly flexible integration approach. However, the emphasis of this type of system is not on the data modelling underlying the database and the conceptual integration in the system is limited; maintaining overall data consistency is difficult. For example, supposing there are two application tools, one requires room volume, the other requires room width, length and height. In a database centred system, all the data would be stored in the database. If the first tool changes the room volume it has no means of knowing how to change the width, length and height correspondingly. Therefore, inconsistent data occurs in the database. In addition, the extension to cover other function is often difficult (Ohsuga, 1989).

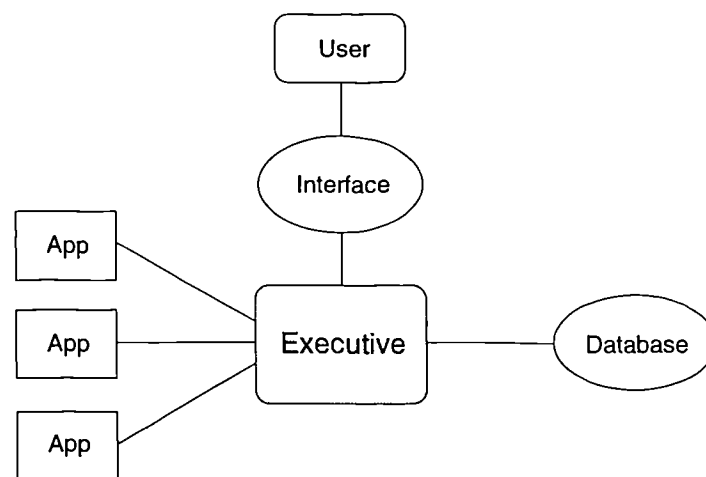


Figure 5.13 Executive-Centred system

An **Executive-centred** approach places emphasis on the management, sharing and synchronisation aspects of computer based design tools (Dupagne, 1991). It recognises that individual tools in a system are not completely separated. Apart from sharing data, they can share common computational resources; several tools can be arranged into a logical sequence for a particular design task. For the purpose of managing the interaction between different tools, the database and the user, an executive is introduced. It controls and synchronises all actions within the system in response to user requests. The executive, together with the database, the user interface and a collection of application tools, form four equally important parts of the system (figure 5.13). This approach implies a need for

process integration and a design process model which describes the sequence of the architects' activities during design. However, discussion in section 3.3 concluded that there is no widely accepted design process model available and it is unlikely that one will be available in the near future (Augenbroe, 1989a).

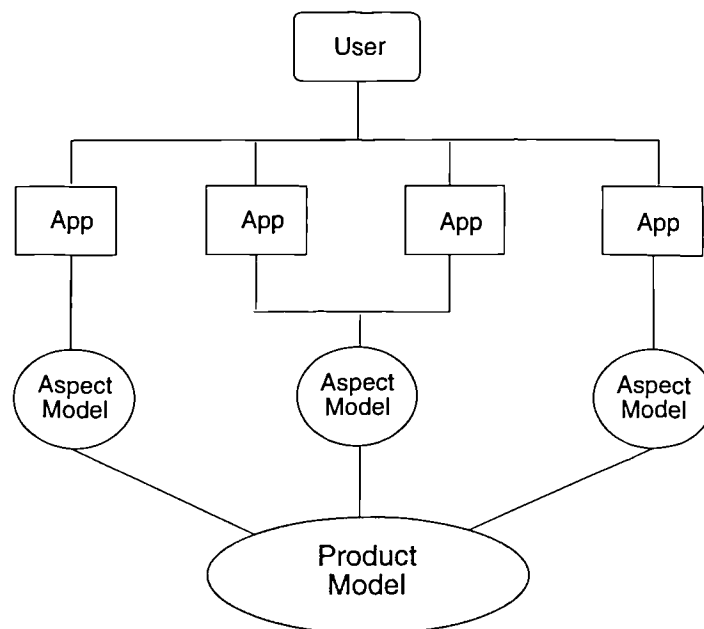


Figure 5.14 Product Model centred system

In the **Product model centred** approach, the system is centred around the objects being designed. The data model in this approach is a description of what the building is rather than how it is used as in the first two approaches. The product model is usually expressed in a neutral format independent from the use of any application tools. However, practically, the "Product model" will always need to be an open model for further extension with its scope defined by the tools it needs to serve, because it is difficult, if possible at all, to define a complete description of the building throughout its life cycle in the short term. Therefore, the concept "Aspect model" is introduced for the product model approach. An "Aspect model" is a partial view of the building object from the view point of one or a set of application tools (figure 5.14). Views of different aspect models are integrated (not simply aggregated) in the central "Product model".

5.4.2 Data exchange and building data modelling

There are two possible approaches to the data exchange problem (Howard, 1989). First, a data exchange interface could be set up between any two tools in the system. By doing so, the data and information exchange between tools could be realised. However, it is an

inefficient approach, since it is extremely difficult for the extension of the system. When a new tool is added to the system, it is necessary to set up exchange links with each and every existing tool.

Another approach towards the data exchange is to separate the data itself from its usage. In this approach, a neutral data description or data model is set up and every tool maintains data exchange with and only with the data model. As figure 5.15 shows it is more efficient in managing the data exchange interface especially when a new tool is introduced to the system.

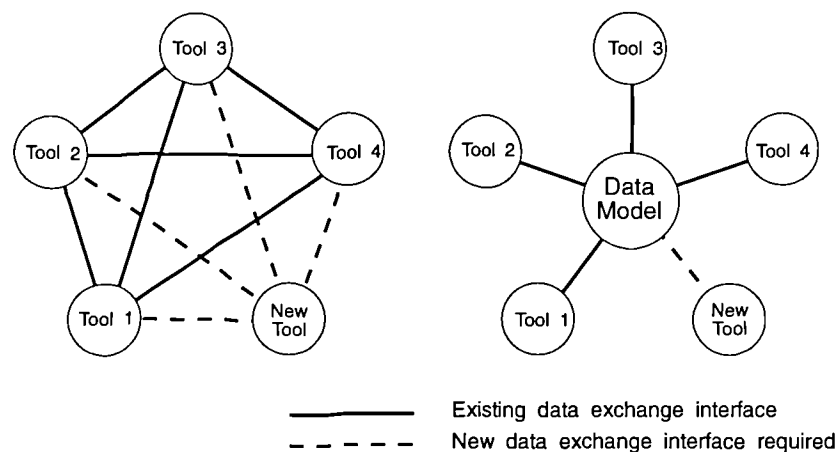


Figure 5.15 Two data exchange approaches

The philosophy of building data modelling is to separate the information structure about the building from the structure of how the information is used and stored (Björk, 1991b). This separation results in two levels of model description, the conceptual level and the physical level. At the conceptual level, the building model contains a logical or semantic description of the data items of the building environment and their interdependencies. At the physical level, the model includes the means by which the conceptual model is implemented (figure 5.16).

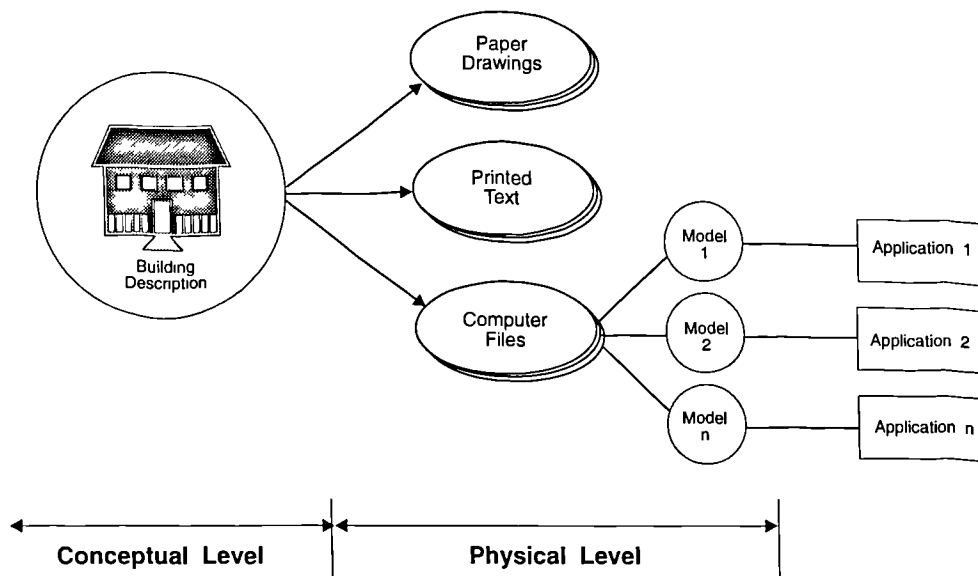


Figure 5.16 Two levels of building data modelling

Traditional architectural models are drawings, i.e., plans, sections, elevations, perspectives, and text specification documents (Schmitt, 1988). The three dimensional building with construction and other related information is only mentally recorded by the designer. This kind of graphic and textual representation is sufficient, though not ideal, if the design is being carried out by one single designer who understands the meanings of all the graphic symbols and text notes and how the different drawings piece together to represent the building. However, difficulties arise when communication is needed with other people. In order to facilitate the communication, all participants must use a common language to describe the building. This principle also applies to computer applications. The task of building data modelling is to defined the building conceptually and explicitly.

Since the mid 1980s, there have been a number of studies into building data modelling (Danner, 1988; Enkovaara, 1988; Björk, 1989). However, no universally accepted building model is available so far. RATAS is a model developed by the Finish building research community (Enkovaara, 1988). It is relatively well documented, and therefore, it is used as an example to illustrate the tasks involved in building data modelling.

The RATAS model is based on the notion of a 'product model' which describes the building using objects or entities and their inter-relationships (Björk, 1989). It adopts an 'abstraction hierarchy' of five levels, *building*, *system*, *sub-system*, *part* and *details*. There are a number of objects at each level of the hierarchy. The totality of the information in the hierarchy is the building data model (figure 5.17).

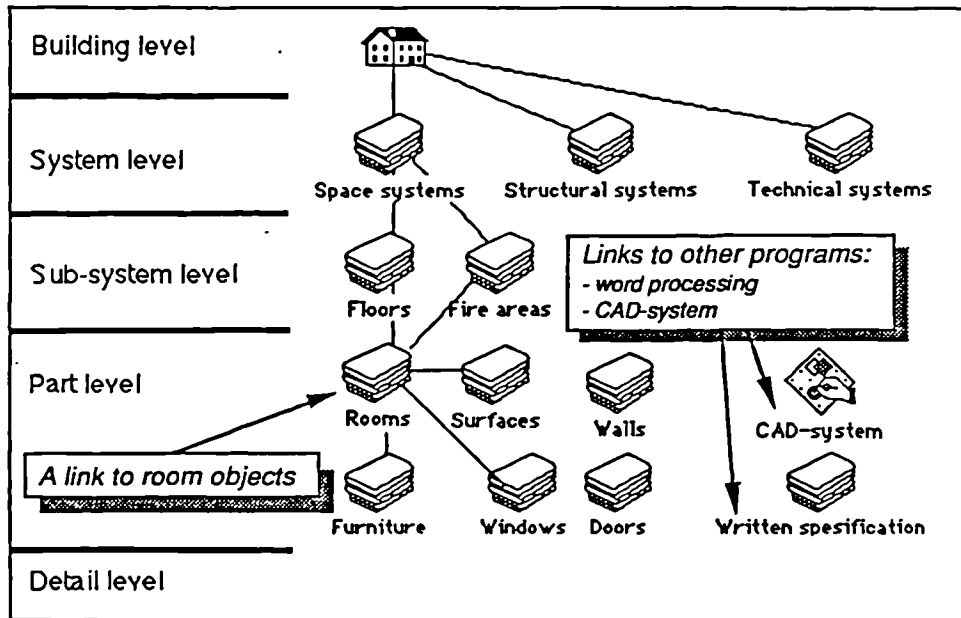


Figure 5.17 Illustration of the RATAS building data model

A data model is not only a catalogue of objects, it contains relationships between objects. The relationship is a description of the real world existence, for example, a room is enclosed by several walls, a ceiling and a floor, a wall is connected to other walls, a window is located on a wall, etc. The RATAS model defines two types of relationship, *part-of* and *connected to*. Data modelling is not to describe a particular building but to provide an abstract framework to describe all buildings. RATAS model uses classes to represent this abstraction (figure 5.18).

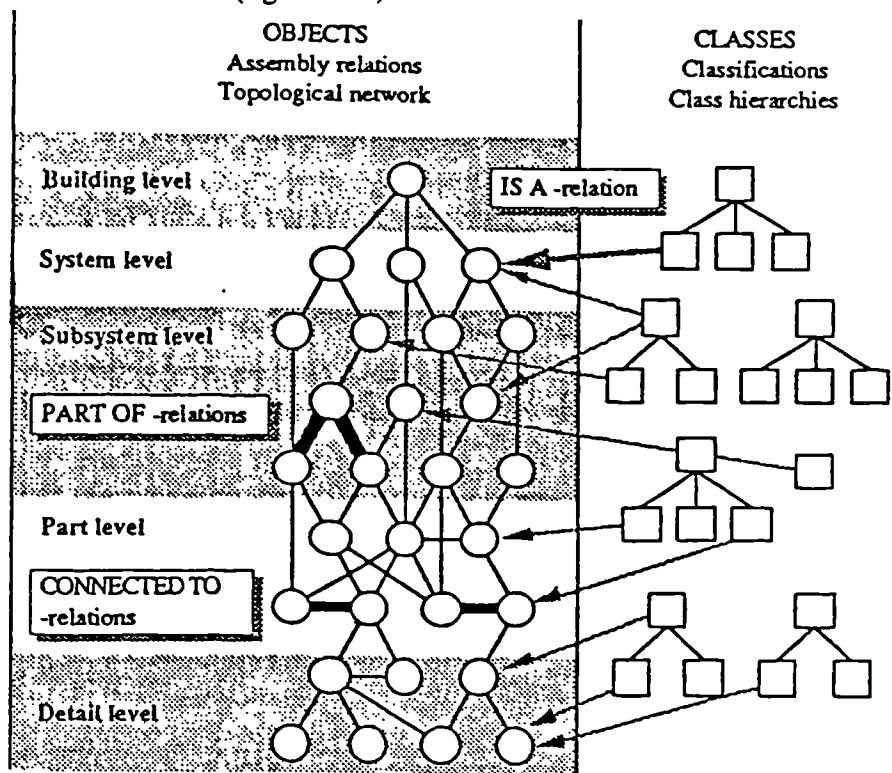


Figure 5.18 The RATAS model class hierarchy and relationships

A class is a generic concept which specifies objects of the same type, i.e., building, room, wall, etc. Each object of that type is called an instance of that class, such as a particular building, a particular room or a particular wall. The RATAS building model is implemented in a form of database and multiple tools can gain access to it (figure 5.19).

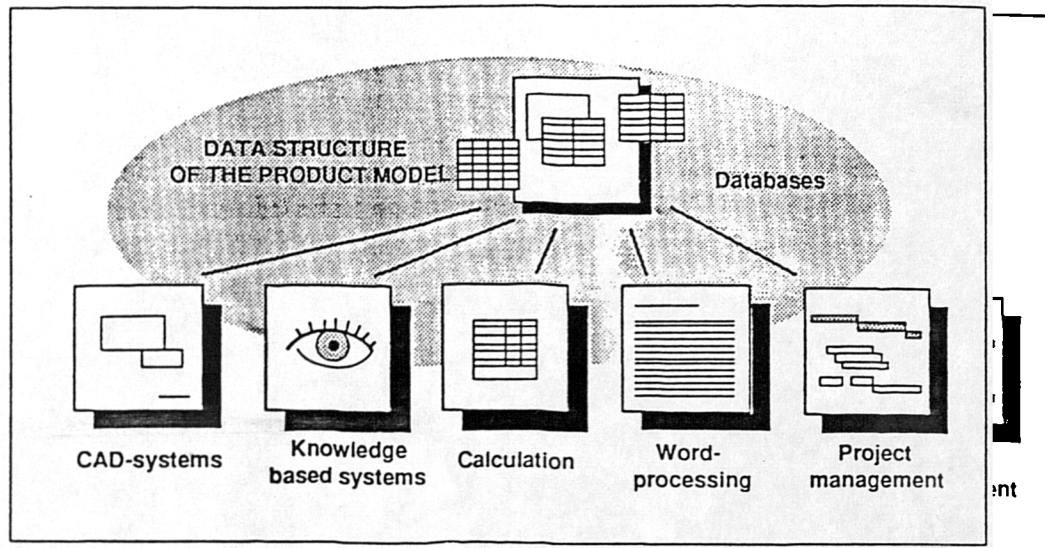


Figure 5.19 The usage of RATAS model

The RATAS model is a very good example of building data modelling. It highlights a number of important issues, e.g., "how to decompose the building into a sensible hierarchy", "how to identify and define building object classes and their relationships", "the extent of details to be modelled" and so on. Björk (1989, 1991b) pointed out that RATAS is still evolving, it is not yet a complete building product model. For instance it only has a very limited coverage of geometrical information.

5.4.3 Data exchange standards

Figure 5.15 shows the importance of using a data model to enable the integration of multiple tools within one system. Another important advantage for using data model is the prospect of wider standardisation. A global data model would enable data exchange not only within a system but also between systems as well.

Historically, the initial requirement for a standardised data model came from the need for different versions of CAD application to share their graphic files. IGES (the Initial Graphics Exchange Specification) of the United States was developed for this purpose (Warthen, 1989). IGES version 1 came out in 1980. It was designed as a drawing oriented standard used for the exchange of technical drawings including textual information and

annotations. IGES was and still is used as a world-wide standard for CAD drawing applications.

However, graphical and geometrical data is only part of the information required in building design. IGES is not able to support the exchange of other type of data such as construction, thermal, light, etc. Therefore a new project, PDES (Product Data Exchange Specification), was proposed in the US to overcome these limitations. In the meantime, similar efforts have been made in other countries, for example, the SET (Standard d'Echange et de Transfert) in France and the VDAFS (Verband der Deutschen Automobilindustrie Flaechen Schnittstelle) in Germany. All these research initiatives have been co-ordinated into a major international program, ISO/STEP. Figure 5.20 shows the evolution history of the data exchange standards (Warthen, 1989).

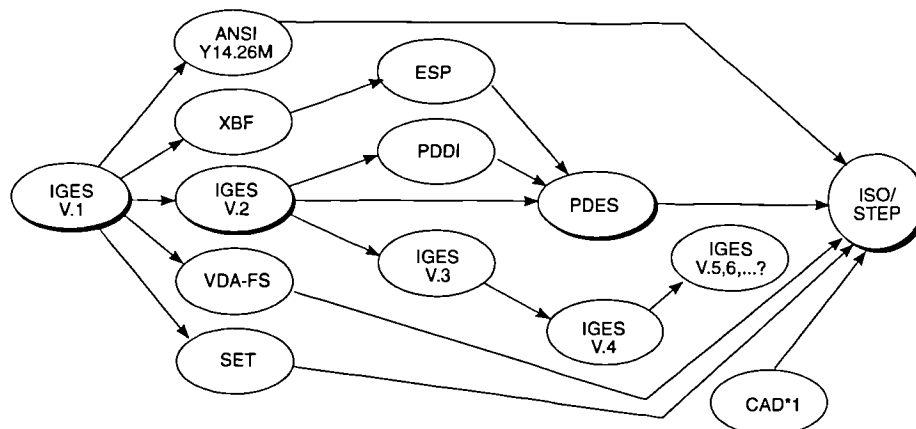


Figure 5.20 The evolution of data exchange standard (after Warthen, 1989)

*"The aim of STEP is to enable the transfer of **ALL** the data associated with the production and use of products. This will include the design, design decisions, manufacturing processes, NC toolpaths, stress calculations, maintenance history and other data associated with a product."* (Harrow, 1991)

STEP covers a very wide application domain. It is not practical nor efficient to take on the whole STEP domain in one project. A layering hierarchical breakdown is adopted in the STEP project (figure 5.21) (Gielsing, 1989).

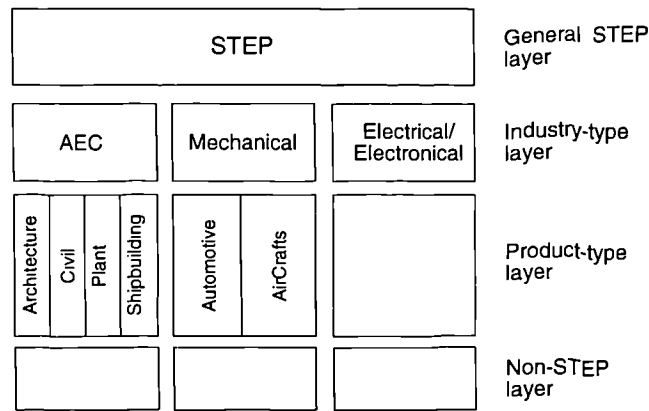


Figure 5.21 Layer structure of STEP (after Gielingh, 1989)

At the top level is the general STEP layer which includes data standards common to all the applications across the STEP domain. The second level or Industry-type layer has three application areas which are divided further into Product-type layer. Moving down a level implies an increase of specialisation. For example, AEC is further divided into Architecture, Civil, Plant and Shipbuilding. There are some data entities which are too specific to be included in the STEP. Another non-STEP layer is proposed for these additional standards. The layer at the lower level defines standards specific to products at that level. It inherits all the standards from the higher level. According to this division, geometry and topology might be included at the *General STEP level*, since they are common to products of all industry fields, while building, room and wall would reside at the *Product-type layer, Architecture*.

Once fully established, the STEP model will act as a reference for the data exchange between application systems (Turner, 1988). The conceptual level is the data exchange reference. It serves as the basis for implementing the physical level and mapping reference for data exchange. As long as system A and B observe the same STEP standard when they describe the same data, their data entities can be exchanged directly or indirectly (figure 5.22).

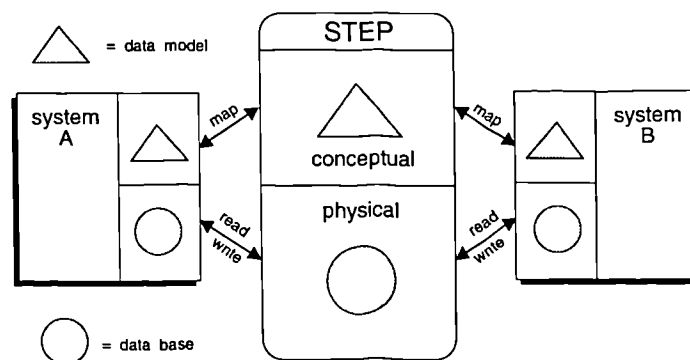


Figure 5.22 Data exchange using STEP (after Turner, 1988)

At present, STEP is still very much a developing research effort; no solid practical standards have yet been finalised. The status of the STEP project can be summarised as follows:

- The whole project is at a stage of testing and proof of concept. There is sometime to go before it becomes a practical standard.
- Developments so far have provided some useful methodologies and formalisms, such as the EXPRESS data modelling language, for software development (Augenbroe, 1991).
- In the near future, any STEP standard will be at the general level; the STEP standard for building system must be considered as a long term goal (Wix, 1991).

5.4.4 Conclusions

The essence of an integrated building design system is that multiple tools in the system maintain data exchange through a common data model. Despite some research efforts such as RATAS, there are still many issues which need to be explored in the development of a building data model.

5.5 Summary

The increasing application of computers in architectural design practice provides a platform to improve architectural design support, and consequently technology transfer. However, existing computer applications have various deficiencies, notably the lack of user friendly interface and the lack of integration between different tools. Recent research has begun to address these problems and integrated building design systems have become an important research subject. Discussion in section 5.4 has concluded that the establishment of a common building data model and the integration of multiple tools through the model are two of the outstanding issues in the development of an integrated building design system. They will be fully explored at the prototype development stage of this study.

CHAPTER 6

PROTOTYPE DEVELOPMENT, ANALYSIS AND SPECIFICATION

6.1 Introduction

The preceding discussion has concluded that the development of Integrated Building Design Systems (IBDS) is one of the keys of improving the architectural design support and technology transfer in the building industry. The essence of such an IBDS is the integration of multiple tools through a common building data model. This chapter starts with an introduction to the development of an IBDS prototype, MultiCAD. The emphasis is on concept proving rather than the development of a commercial software product. This chapter concentrates on the analysis and specification aspects of the prototyping process. Major issues include:

- the background of the prototype, the COMBINE project,
- domain of the prototype and design request analysis,
- functional requirements in order to meet the perceived design requests,
- theoretical calculation methods to be used to satisfy the design requests, and
- data and process analysis.

6.2 Software Development Process

The development of a computer program is a software engineering process. The IEEE Standard Glossary of Software Engineering terminology (IEEE, 1983) defines software engineering as: "The systematic approach to the development, operation, maintenance, and retirement of software," where "software" is defined as: "Computer programs, procedures, rules, and possibly associated documentation and data pertaining to the operation of a computer system." Figure 6.1 shows the classical "waterfall" diagram showing the life cycle of the software engineering process.

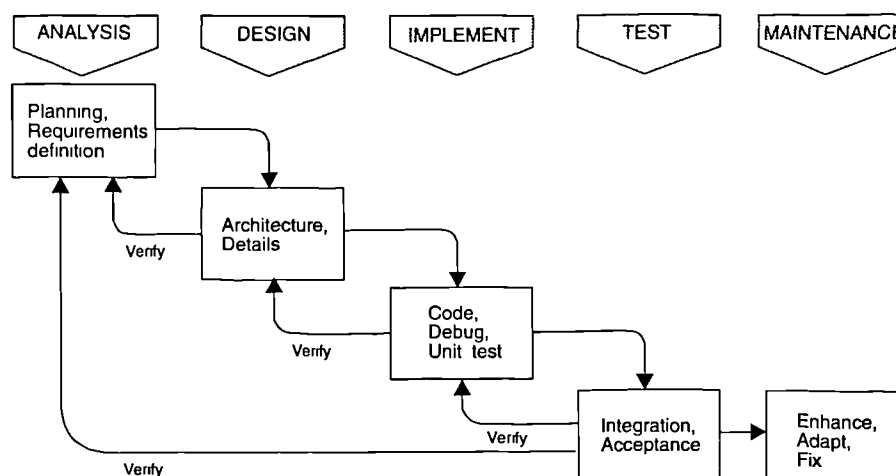


Figure 6.1 Software development process and tasks (after Fairley, 1985)

The software engineering process consists of five stages or phases, **Analysis, Design, Implement, Test and Maintenance**; the tasks of each stage are summarised as follows:

- The **Analysis** phase includes two tasks, project planning and requirement definition. In order to ensure the successful conduct of software development, a comprehensive work plan has to be developed at the beginning of the project. Issues to be studied at this stage include the scope of the work to be done; the resources required; the tasks to be accomplished; the milestone to be tracked; the schedule to be followed and so on (Pressman, 1987). It is important to thoroughly examine the problem domain in order to identify correctly the functional requirements. Following the initial plan, the next task is to transfer the functional requirements into a software functional specification which defines what the software will do. This is a vital task because a poorly specified program will not be attractive to its users no matter how well-designed and well-coded it might be.
- The **Design** phase is concerned with identifying software components (functions, data streams and data stores, etc.), specifying relationship among these components and providing a blueprint for implementation (Fairley, 1985). It is a process of designing a software product which can satisfy the requirements identified in the proceeding stage. The design phase is usually divided into two sub-tasks, Architectural design and Detailed design. Architectural design identifies software components, decoupling and decomposing them into process modules and conceptual data structures. Detailed design further structures the system into details. The output of the design phase is a system structure which is ready to be transferred into programming code.
- The **Implementation** phase consists of the translation of the system design specification into source code, debugging, documentation as well as unit testing. During implementation, cyclic activities of code reviews, walkthroughs and inspections are conducted in order to verify the completeness, consistency and suitability of the software product. At the end of this phase, a running version of the software is available but which is not ready to be delivered to the users.
- The **Test** is the last process before the release of the software. It acts as a maximum guarantee of the quality of the final software product. It normally involves two activities, integration testing and acceptance testing. The integration test aims at ensuring that the system operates correctly as a whole. The acceptance test aims at demonstrating the satisfaction of the system requirements set out at the **analysis** phase as well as with real users' needs.
- The **Maintenance** phase begins after the satisfactory testing and the subsequent release of the software. The tasks for the software developers, at this stage, are the

enhancement of system capacities, the adaptation of the software to new computing environments as well as the correction of bugs detected during its usage.

Prototyping is a technique often used in the software development process. It enables the software developer to create a model of the software to be built. The benefit of using this technique is to further test the underlying concepts and specification of the software product against the real needs of the user before committing large scale resources. Prototypes can take on several levels of complexity. On one hand, they can be a simple paper mock-up of the human-machine interaction which enables the user to understand how the interaction will occur. On the other hand, a prototype could be a full working version of the software only with less strict requirements on the system performance, documentation and so on (Pressman, 1987).

MultiCAD is a working version prototype which is used to investigate issues involved in the development of an integrated building design system. Because it is only a prototype, the software development cycle described above will be used only as a guide not all the tasks are necessarily carried out. This chapter is on the **Analysis** phase, chapter 7 is about **Design** and **Implementation** and chapter 8 deals with **Test**.

6.3 Concepts of COMBINE

MultiCAD has been developed in the context of the ongoing European COMBINE project, consequently the concepts and the background of this project are introduced below.

The overall objective of the COMBINE project is the test of concept for a full integration of energy-related Building Performance Evaluation (BPE) tools into the building design process. The initiative was based on the following observations:

- There are many discrete BPE tools available but they are not used effectively by designers in practice.
- A full description of the design object or design data, such as building, is not available to a wide range of BPE tools.
- There is a gap between design requests and BPE instructions
- A gap also exists between the BPE simulation results and design context performance tasks.

COMBINE is proposed for the purpose of exploring ways to bridge these gaps. It emphasises data exchange between different tools and the need for a unified building data model (Augenbroe 1989a, 1989b). Figure 6.2 shows the structure of the COMBINE project.

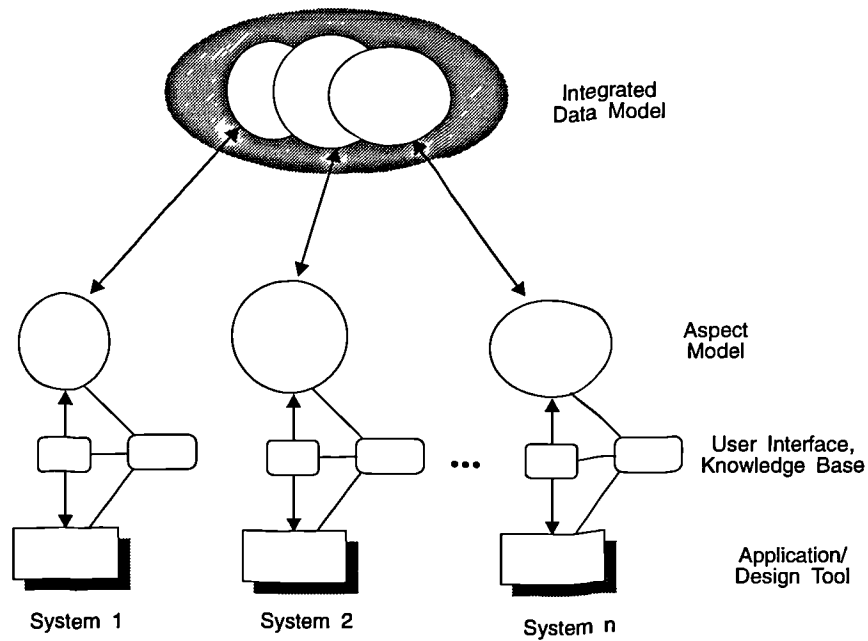


Figure 6.2 Structure of COMBINE project

At the core of COMBINE is the Integrated Data Model (IDM) which is a data model containing a full description of the design object [This is a long term target]. Sitting around the IDM is a collection of subsystems or Design Tools (DTs). Each DT contains one or several BPE tools, and maintains an Aspect Model which represents a limited set of generic data about the design object from view point of that DT. The aspect models for all the subsystems can eventually be mapped and transferred to the IDM. Definitions of IDM, Aspect Model, DT and BPE tool will be given in the following sections.

Augenbroe (1989a), one of the initiators of the COMBINE project, described architectural design as a three dimensional integrated process (figure 6.3). The time axis describes the design stages, i.e., brief, outline design, scheme design, detail design and so on. The domain axis represents the individual areas of design or design sub processes, such as structural design, lighting design, interior design. The aspect axis is for the criteria or goals, i.e., costing, comfort, energy consumption, etc., on which the design will be evaluated.

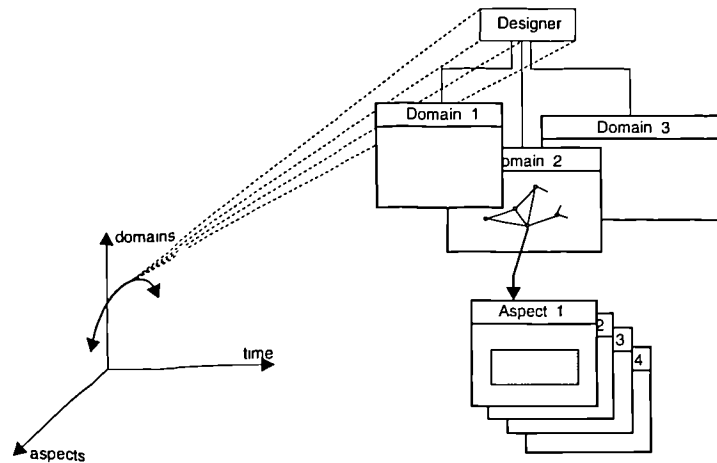


Figure 6.3 Building life space and scheme of design support

Building design is not sequential process, nor is it conducted by a single person (discussion in Chapter 3). A typical design process usually involves several participants working together, i.e., architects, civil engineers and other experts. Each expert is responsible for certain tasks. At present, the communication between them relies heavily on personal contact. A full integrated building design system should be able to handle the exchange of information between different tasks. The COMBINE project is a step toward this long term goal.

6.3.1 Data integration and process integration

Augenbroe (1988) has identified two areas of integration, which have different implications for IBDS research.

- *Data integration: research in this area will lead to a standard for describing design objects and methods for making object descriptions available through a neutral format to different design domains, and within each domain, to different design aspects.*
- *Process integration: this involves definition of the design context for any aspect-related task, such as performance evaluation, and handling the flow of information and decisions between design domains, and between designers."*

Data integration requires making information available to all Building Performance Evaluation tools when needed. The data integration along the three dimensions is described by Augenbroe (1988) as follows:

- Along design phase axis, the volume of data grows incrementally as the design proceeds. Existing data can be changed at later stage if designers decide to make changes on their previous design decisions.

- Along the domain axis, each domain usually has its own conceptual perception of the design object. As the design progresses in each domain, related design elements will be gradually refined where appropriate. Any change of data in one domain must be reflected in the other domains.
- Along the aspect axis, different aspects of building design have to be assessed separately as well as in relation to others. During the evaluation process, the data input and output must be maintained coherently.

The key to data integration is data modelling which provides a data description independent from any specific use of the data (Warthen, 1989). Arguments for the need for building data modelling have been presented in section 5.4 of Chapter 5.

Process integration also known as task integration deals with questions like "when should a design request be made?", "what decision should be made based on the response to the request?" and "what to do next?" Obviously, data integration is the pre-condition of process integration. Process integration involves the design methodological issues, i.e., a design process model. It has been argued in section 3.3 that this is unlikely to be achieved in the foreseeable future. Given the absence of a design process model, the design sequence must remain under the designer's control and design support systems need to be flexible and responsive to designer's requests.

6.3.2 Building performance evaluation tools and design tools

As the name suggests, a Building Performance Evaluation (BPE) tool is used for the evaluation of the building performance in a specific aspect, such as thermal, lighting, acoustic performance and so on. They deal with the design object from the view point of one specific aspect of building design not from the view point of an integrated design context. For example, the BPE tool for the calculation of daylight factor will not view a room as "an office room where people are working, but as a space with an opening(s) which allows daylight to come in and several interior surfaces of given light reflectance." In practice, BPE tools are often operated by domain specialists. Daylight tools are operated by lighting consultants; structural tools are operated by structural engineers and so on. These specialists are responsible for the interface between design contexts and the BPE process, such as transferring the design objects into input data and translating the output results into design oriented information

Design Tools (DTs) differ from BPE tools in a sense that they handle design context and provide (some) local intelligence for specific domain-oriented design tasks (Dubois, 1991) while the BPE tools do not. The implication of local intelligence includes bridging the gap between design context and BPE context, which covers several information translation

tasks, i.e., translating design requests into BPE requests, translating the design object descriptions into BPE input as well as interpreting the BPE results into design information. These tasks are usually conducted by specialist experts in the use of independent BPE tools.

For the design tools of COMBINE, it is intended that these translations will be done by the system. For example in the average daylight factor calculation, the BPE tool is the calculation method itself, which reads $DF = (TW\theta)/(A(1-R))$. In this equation " T is the diffuse transmittance of glazing material including effects of dirt, W is the net glazed area of window (m^2), A is the total area of interior surface (m^2) (ceiling, floor, windows and walls), R is the area-weighted average reflectance of interior surfaces (ceiling and floor and walls, including windows), and θ is the angle in degrees subtended, in the vertical plane normal to the window, by sky visible from the centre of the window." (CIBSE, 1987) In its operation, the BPE tool requires input of T , W , θ , A , and R , and produces a result, e.g., 2.0%.

However, an ordinary designer may have difficulties in understanding the meaning of the input data and in interpreting the evaluation result. Therefore, a design tool should ask for input data in a way which is familiar to designers, such as area of each interior surface and their finish materials, window opening area, window frame type, glazing type, etc. The translation from these architectural objects into BPE input should be done by the design tool itself. In addition, the design tool should provide an explanation of the calculation result, in this example, judgements could be given on whether the daylight level, 2.0% meets the relevant standard.

A design tool can consist of one or more BPE tools. Figure 6.4 illustrates the operation of a design tool and its relation with BPE-tools, which can be described as follows (Augenbroe, 1988). The architect works on the building in a particular design context. He/she deals with objects such as building, rooms, walls, etc. Suppose at one stage, the architect needs to evaluate the daylight performance of a particular room. He will issue a design request through some kind of interface control. Behind the user interface, several translations take place from the design context to BPE context. The design object, room, is translated into model data which contains interior areas, surface light reflectances, opening areas, etc., the design request is translated into process data which invokes a sequence of calculations, such as area weighted light reflectance, total interior area, and so on. At the end of the sequence, a result is produced, e.g., 2.0%. The design tool then translates the result into meaningful design information, say the daylight level of the room under investigation satisfies the CIBSE standard for a particular activity.

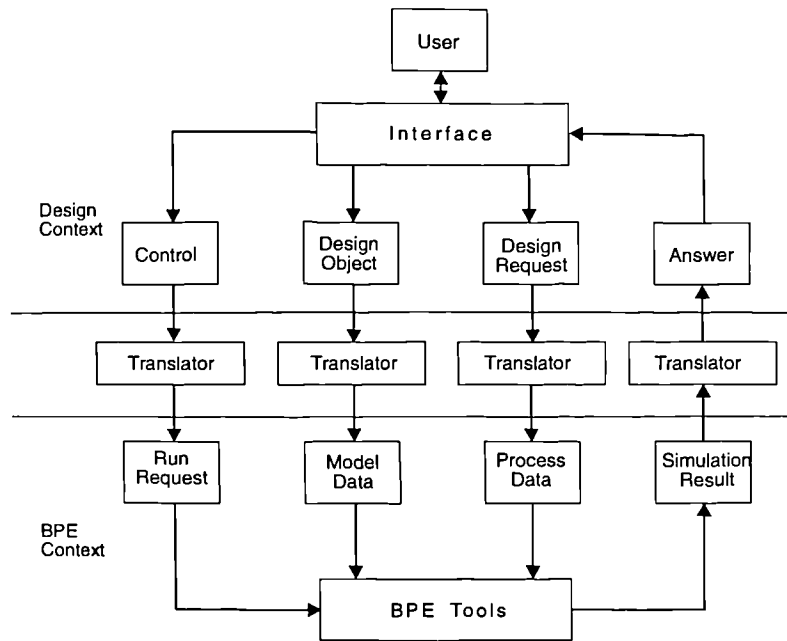


Figure 6.4 The operation of a design tool

The current COMBINE project includes the following six DTs for different domains:

- DT-1 Construction design of external building elements.
- DT-2 HVAC design.
- DT-3 Dimensioning and functional organisation of inner spaces.
- DT-4 Input generation for thermal simulation tool in the later design stage.
- DT-5 "L-T method" in the early design stage.
- DT-6 Energy-economic design.

The six DTs are essentially stand-alone systems; each is considered as an actor or a participant in the design process in the context of COMBINE. It represents a particular view of the design participant. For example, DT-2 reflects the view of a HVAC engineer, DT-6 is the view of economic consultants and the rest are more or less the views of architects and/or engineers. The intention is that the six DTs would work together in a similar way as real world design participants in a design team. Each of them has a specified domain of the design task, and has one or more BPE tools. They have their own view on the design objects (aspect model) and finally they should be able to work together through the IDM.

6.3.3 Aspect model and integrated data model

An aspect model is the conceptual schema of the design object, i.e., the building itself or parts of the building, from the view point of a DT in COMBINE. It defines the way in

which the DT perceives the design context. The view difference of each DT can be understood by the following comparison. DT-1 which is concerned with external element design, would naturally perceive the building as consisting external elements with construction properties. On the other hand, DT-2 which is about HVAC design, might see the building as a collection of internal spaces of various sizes and their relationships. The set of data which describes a particular DT's view of the building is the Aspect Model of that DT.

At the core of COMBINE is the Integrated Data Model (IDM). The IDM is the general conceptual schema for the design object description, it is used to support all data exchanges between DTs in the project. Figure 6.5 illustrates communications between DTs through the IDM.

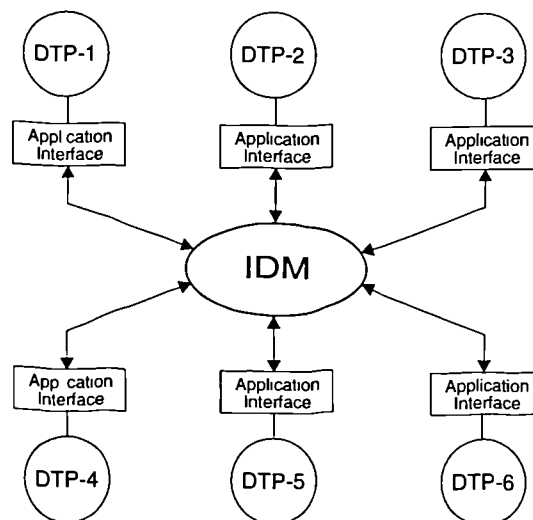


Figure 6.5 IDM and DTs in COMBINE

During operation, a DT extracts data from the IDM through the frame of its *aspect model*, performs some design tasks on the aspect model data and returns appropriate data back to the IDM. The data mapping and translation activities are handled by the *application interface* as indicated in figure 6.5. The *application interface* is implemented as an EXPRESS/STEP interface kit (Plokker, 1991). The emerging STEP international standard has been discussed in section 5.4.3. EXPRESS is a conceptual schema language (Spiby, 1991) which provides rules to specify a data definition unambiguously with the constraints on that data clearly and consistently stated.

The scope of the IDM, in relation to the aspect models of DTs, is an issue for debate. There are two views, (1) the IDM contains all the data concerning the building; (2) the IDM only contains data shared by two or more DTs. At this stage, the scope of the IDM does not cover all data items required by every DT, but only those shared by more than one

DT. Data specific to only one DT is stored locally since no other DT requires that particular data item. However, the conceptual structure of the IDM must be broad enough to support all the DTs' aspect models. Any future extension of modelling scope should be a matter of growing on the basis of the existing model and should not cause the total abandonment of the model (NCL, 1992).

6.4 Domain of the Prototype (MultiCAD)

This study is about the development of DT-1, called MultiCAD, the prototype for *The construction design of external building elements* (Wiltshire, 1989).

The pre-defined domain of the COMBINE project is energy-related issues in domestic and non-domestic building design. MultiCAD investigates the specification of building external elements at the early design stages. The importance and existing problems in the early stage of the building design process were highlighted in the discussion of Chapters 3 and 4. The objective of MultiCAD is to explore possible improvements for design support by developing an integrated building design system.

6.4.1 Targeted users

At the early stage of building design, several participants are involved in the decision making process, such as planners, consultants, engineers and architects. These participants have their own perceptions of the design problem and particular needs for support. The intended target audience of MultiCAD is the architect.

This choice was made based on the conclusions of the analysis presented earlier.

- Architects have the most influence on the performance of the building (section 2.5).
- The early design stage is the most crucial phase effecting the final performance of the building (section 3.5).

The immediate objective of MultiCAD is to test the feasibility and concepts of an integrated design support system. The aim is to provide the basis for developing an effective design support tool for architects in practice in the longer term and contributing to the general improvement of technology transfer in the building industry.

6.4.2 Use scenarios

The analysis presented in section 3.3 and 3.4 indicated that most architects adopt an interactive thinking and working style. The requirements for the computer based design support system have emerged from the discussions in chapter 4 and chapter 5. The

following fictional story illustrates how MultiCAD can better meet the architects' requirements.



Figure 6.6 Architect at work

Jim Smith, a senior architect at Smith & Partner architectural design office, sits in front of his PC and is ready to work. The time is January 1995. He received a commission for the design of a house last week. Jim has already drawn some rough sketches of the house layout on a piece of paper.

From the *Window Manager* of his PC, Jim clicks the Icon for MultiCAD which is a package for the early design stage, developed by the Department of Architecture at the Newcastle University. MultiCAD opens with a *CAD Window*, through which Jim puts draft drawings on the screen very quickly together with some design parameters, such as wall, roof and floor constructions. Then Jim opens three small *Windows* that show him the elevation, section and perspective views of the house. He is not very happy about the position of a window shown on the elevation view. The *CAD Window* allows him to simply drag that window around to a new position of his choice. The change is reflected dynamically in all the view *Windows*.

Jim is fairly happy with the appearance of the house. His next thought is to get an idea about the expected energy consumption of the house, as energy conservation has become one of the most important issues in building design since mid 70s. Jim chooses the *Energy* item on the *CAD Window* and a graphic report on the energy consumption is presented. This calculation is done using the existing information supplied by Jim and default data provided by the system. In a text *Window* Jim gets a full report of the default data. He decides to make changes to the construction type of some of the building elements. He selects the elements, then chooses the *Construction* menu. A list of typical construction types is shown. Jim can choose

one from the list or can define a new construction type. After the changes are accepted, a new calculation is carried out on the building energy consumption, and a graphical report is given. In addition, a comparison is available between the results before and after the change.

In order to reduce the heat loss, Jim has specified a very small window for the northerly facing kitchen. He worries whether the kitchen meets the daylight requirement of the CIBSE guide. From the *CAD Window* Jim selects the kitchen, then the *Daylight* option on the menu. A *Daylight Window* opens up for the kitchen, which shows the current data related to the room, and the result of the daylight calculation for the kitchen. The calculated average daylight factor for kitchen is 1.0%. The standard requirement for kitchen is 2.0% which is indicated on the *Daylight Window*.

Jim wants some helps on what to do next. He opens the help window which includes all the information contained in the CIBSE window design guides. He learns how to increase the daylight in the room. First, Jim tries to change the internal finishes to lighter colour, but he still cannot get the required daylight factor. Then he chooses to calculate the window area, he simply sets the required daylight factor and window area needed to satisfy the daylight requirement shows instantly. Jim decides to accept the new window area. The system prompts him that the change in window area will have an effect on the building energy consumption and Building compliance with the Building Regulation. He checks the impact of the change on these two aspects before making the final decision on the window.

In a similar easy and interactive fashion, Jim checks the possible overheating problem for the southerly facing living room.

Finally, Jim produces a documented report of his design by choosing file print and specifies the information he wants to include. The outputs include drawings and specification of the design, as well as graphical and text reports on the building evaluation in the aspects of energy consumption, building compliance, daylight, overheating and so on.

6.4.3 Design context

Wiltshire and Lockley (1989) defined the design context of MultiCAD as being

"that of an actor, principally an architect, involved in the specification of the construction of the external building elements (floors, walls and roofs). This activity is centred on the thermal performance of the elements. The design

stage is envisaged between outline proposal and scheme design of the RIBA plan of work. Here the generic form of the element and material properties may be specified but not necessarily the proprietary products from which the building will be built".

There are many factors which have to be considered during the specification task of the building external elements. Among these factors, some are external constraints which are beyond designer's control, i.e., client's requirements, planning legislation, etc. The designer has to work within these constraints and comply with their requirements. Other influential factors are requirements of design criteria. These are aspects on which the quality of a building or a building design is judged. The following is a list of the major factors in these two categories, which have to be considered at the early design stage:

External Constraints:

- client's requirements
- planning legislation
- **Building Regulation**
- site restriction
- climate conditions

Design Criteria:

- aesthetic
- **thermal**
- acoustic
- **light**
- fire
- buildability
- durability
- security
- cost

As stated above, the pre-defined domain of COMBINE project is with energy-oriented issues. Therefore, MultiCAD as one of its prototypes also concentrates on the energy-oriented tasks. Among the influential factors listed above, the Building Regulations (Part L in particular), thermal and light have direct implications on building energy consumption. Therefore, they are covered by MultiCAD. Climate conditions also have an impact on the building energy performance. However, it is not considered as a separate domain entity. Its influence is reflected in thermal and light. For thermal and light aspects, several tasks can be identified which have impacts on the design context of MultiCAD.

- Building energy consumption. It is a major consideration of specification tasks at the early design stage, such as deciding on window area, and construction of building elements.

- Room daylight. The daylight impact at the early design stage is primarily on the window opening area of a room, the specification of window frame and glazing type as well as the internal finishes of the walls, floor, ceiling, etc.
- Room overheating. Overheating has to be considered when deciding window opening area and glazing type, especially for southerly facing rooms.
- Elemental heat loss. Heat loss, in particular heat loss rate or U-value is a widely accepted measure of the thermal performance a building element.

These four aspects, together with the Building Regulation compliance (part L), form the domain of MultiCAD (table 6.1).

Table 6.1 Domain of the prototype

Level	Building	Room	Element
Tasks	<ul style="list-style-type: none"> • Energy consumption. • Compliance with Building Regulation Part L. 	<ul style="list-style-type: none"> • Daylight • Overheating 	<ul style="list-style-type: none"> • Heat loss, U-value

The five areas covered in MultiCAD deal with different components of the building. Energy consumption is a performance criteria of the building as a whole. Daylight and Overheating are at the room level, while Heat loss is for individual elements. This difference is shown in the layout table 6.1. In addition, any design decision for one design aspect has impacts on the others. For example, Daylight might require an increase of window opening area for a room. As a result, the building energy consumption might be increased if the room is northerly facing, or it might cause overheating problem if the room is southerly facing. Therefore, these aspects have to be considered in an integrated design context.

6.5 Design Request Analysis

The objective of the design request analysis is to identify the designers' needs and the functional requirements of MultiCAD. It has to be stated that MultiCAD will not seek to answer all the possible design requests during the design process. Its main objective is to demonstrate the principle of an integrated design support system and its advantages over traditional discrete programs.

It is impossible to anticipate all design requests under all possible individual circumstances. Nevertheless, several generic types of design request have been identified through an analysis of some most common design requests for the design aspects covered

in this prototype (table 2). This table is arranged in a design object/aspect matrix. It means that the designer needs to make decisions on building objects while considering the impacts of/on design aspects. For example, at one stage, a designer has to decide window area where daylight is one of the considerations. The requests in this case might be:

- What is the required window area for a room in order to have a daylight factor of XX?
- What is the impact on daylight if I change the window area to YY?

These requests are listed in window/daylight box in table 2. Other requests are arranged based on the same principle. It has to be pointed out that table 2 is not an exhaustive list of all the design requests, nor does it imply any sequence in which the requests might be raised.

Table 6.2 Analysis of typical design requests in energy, compliance, daylight, overheating and U-value

ASPECT OBJECT	ENERGY	COMPLIANCE	OVERHEATING	DAYLIGHT	U-VALUE
Building	Annual energy consumption? Measures to improve the thermal performance of the building? Total heat loss rate? What is the impact on energy consumption if the building orientation is changed?	Do the thermal properties of the building comply with Part L of the Building Regulations?			
Room	What is the heating requirement of the room?		Does the room overheat? What is the peak summer time temperature within the room? What steps can be taken to avoid overheating?	What is the daylight requirement of the room? Is the daylighting level satisfactory? Is the room too deep?	
opaque Element	What is the U-value of the element? What is the U-value required to limit heat loss to XX?	U-value for an element in order to comply with Building Regulations?		What is the impact on the daylight factor if the interior element surface condition are changed? What kind of internal finish can I use to improve the daylight?	What is the U-value of the element? What is the impact on the U-value if I change the material or/and thickness of a layer? What is the thickness for the insulation should I use to get a U-value of XX?
Window	Heat loss rate through windows? Impact of window area and U-value on energy consumption?	Maximum window area to comply with Building Regulations? What is the proportion for double glazing required if total window area exceeds the Standard?	Maximum window area to limit overheating problem? The use of shading facilities to prevent overheating?	What is the required window area for a room in order to have a daylight factor XX? What is the impact on daylight if I change the window area to YY? Do window shade and position have any impact on daylight?	What is the U-value of the window? What will the U-value be if I change the frame type or/and glazing type?

For each design aspect, the designer, first of all, needs general information such as Building Regulation's requirements or related standards. For example, in the daylight aspect, the designer would want to know the authoritative daylight requirements for a room.

Then the designer would like to check his design against the requirement. This involves the calculation of the daylight level in a room and the comparison with related standards.

Another type of request is a "what if" query. In other words, the designer changes one or more design parameters, such as the window area or internal surface type, and he wants to know the impact of the design change on the daylight level.

In some circumstances, the designer may wish to achieve a value of a design parameter in order to meet a design requirement. For example, the designer may wish to set the north facing window to a minimum size in order to reduce the heat losses. On the other hand, the daylight requirement demands an adequate window size. Consequently, the designer might want to know "what is the minimum window size that satisfies the daylight requirement".

Based on the above analysis, the following generic types of design request can be summarised:

- (1) general guidance on a design aspect,
- (2) evaluation of a design solution against specific criteria,
- (3) sensitivity test, examine the impact of a parameter(s) on the result,
- (4) value of a design parameter to meet a design target.

General guidance provides design-oriented advice concerning specific design tasks. Evaluation support provides the means to evaluate design decisions. The sensitivity test enables designers to test the impact on the designed performance due to changes in a parameter. The fourth type of request is to enable designers directly obtain a value for a design parameter, i.e., window area for a room, which meets the design requirements. Taking room daylight calculation as an example, the design requests would be as follows:

- General guidance on daylight design of a room?
- What is the average daylight factor of this room?
- What the average daylight factor for the room will be if a parameter(s) (i.e. window area, reflectance of the wall surface) is/are changed?
- What is the window area needed to achieve a required average daylight factor for the room?

The interactive nature of the designers' requests must be emphasised and the requests must be capable of being raised at any time in any sequence. This requirement is reflected in the development of MultiCAD.

6.6 System Functional Requirements

The aim of a functional requirement specification is to define the tasks that the prototype has to perform in order to meet user requirements. The functions of the prototype reflect the design requests discussed above, and therefore, are arranged in a similar format to the design request analysis.

The functional requirements presented below are divided into several sections. Section 6.6.1 is about general requirement on the development of MultiCAD, which is set out based on the analysis of architectural design and the integrated building design system. Following it are the functional requirements for each aspect included in the prototype. These requirements specify the tasks that MultiCAD needs to perform in order to effectively help architects during specification of external building elements.

6.6.1 General requirements

- (1) The prototype will be developed based on existing authoritative BPE methods.
- (2) The BPE methods selected should be reasonably easy to handle, while comprehensive enough to demonstrate the COMBINE philosophy.
- (3) The prototype should enable multiple tools to run concurrently and enable the user to operate in an interactive manner.
- (4) The dynamic information exchange among different tools within the prototype should be maximised in order to promote the interaction, integration and user friendliness of the system.

6.6.2 Building energy consumption

- (1) The prototype should be able to provide general guidance on energy efficient design measures.
- (2) The prototype should be able to provide reports, both graphical and numerical, on:
 - annual and monthly energy consumption.
 - annual and monthly heat losses through building elements and ventilation.
- (3) The prototype should be able to evaluate the impact on annual energy consumption of changes of any one or a combination of relevant design parameters. The most common parameters include:
 - building orientation,
 - elemental U-value,
 - window area,
 - window U-value,
 - heating temperature demand,

- (4) The prototype should be able to suggest the required value for the following design parameters in order to satisfy energy consumption target.
- elemental U-value,
 - window area,
 - window U-value.

6.6.3 Room daylight

- (1) The prototype should be able to provide general guidance on daylight.
- (2) The prototype should be able to calculate the average daylight factor of a room.
- (3) The prototype should be able to check the adequacy of daylight distribution within a room.
- (4) The prototype should be able to evaluate the impact on daylight due to changes in one, or a combination, of the following parameters:
- window area,
 - glazing type,
 - reflectance of internal surfaces.
- (5) The prototype should be able to calculate the minimum window area for a specified room to satisfy its daylight requirement.

6.6.4 Building element heat loss

- (1) The prototype should be able to provide general guidance on building elemental thermal performance.
- (2) The prototype should be able to calculate the steady state U-value of a specified element.
- (3) The prototype should be able to evaluate the impact on the U-value of an element due to changes in one, or a combination, of the following parameters:
- material of a layer, include insulation layer,
 - thickness of a layer, include insulation layer,
 - surface exposure conditions.
- (4) The prototype should be able to calculate the value of the following parameters for a required U-value.
- thickness of a layer,
 - thermal resistance of a layer.

6.6.5 Compliance checking

- (1) The prototype should be able to provide information on the statutory requirements of the Building Regulation (England and Wales) Part L.

- (2) The prototype should be able to check the compliance of a building with Part L of the Building Regulation.
- (3) The prototype should be able to evaluate the impact on a building's compliance with Part L, due to changes in one, or a combination, of the following parameters:
 - window area,
 - window U-value,
 - area of building element,
 - elemental U-value.
- (4) The prototype should be able to calculate the value of the following parameters, which enables the building to comply with the Building Regulations Part L.
 - window area,
 - window U-value,
 - U-value of building element.

6.6.6 Overheating risk

- (1) The prototype should be able to provide guidance on measures to avoid overheating.
- (2) The prototype should be able to assess the overheating risk of a room by calculating its peak summer time temperature.
- (3) The prototype should be able to assess the impact on the overheating problem for a room, due to changes in one, or a combination, of the following parameters:
 - window area,
 - window orientation,
 - glazing type,
 - construction of room elements,
 - incidental heat gain sources in the room.
- (4) The prototype should be able to calculate the value of the following parameter in order to avoid overheating problem in the room.
 - window area.

6.7 Model Identification

The two preceding sections specified the design requests the prototype intends to support and the functional requirements of the prototype in order to support these requests. This section introduces the theoretical models which provide the adequate BPE functions, they include:

BREDEM-8 (Anderson, 1989),
CIBSE average daylight factor (CIBSE, 1987),
CIBSE standard U-value (CIBSE Guide Section A3),

Compliance checking with the Building Regulations Part L (Building Regulation 1990 Part L),
CIBSE summertime temperature (CIBSE Guide Section A8).

The calculation process of each method were documented in DT-1 progress report (Lockley, et al, 1991). In the following, only a very brief introduction is given for each method. More details can be found in either the original sources or the DT-1 progress report mentioned above.

6.7.1 BREDEM-8

BREDEM (Building Research Establishment Domestic Energy Model) is a basic model for energy consumption in domestic buildings, which was developed by the Building Research Establishment. BREDEM has several versions, each of which has particular aims.

BREDEM-8 (Anderson, 1989) is used in this prototype. It is a two-zone model. Zone 1 is the living area, normally including lounge, living room, sitting room and dining room. Zone 2 is the rest of the house. By such a division, the zones can be considered separately in terms of heating demand, for example specifying different heating schedules or/and different heating temperatures.

The overall building energy consumption depends on factors such as the size of the building, heating zone division, elemental construction, window area, frame and glazing type, heating temperature requirement, etc.

It is acknowledged that BREDEM-8 does not apply to non-domestic buildings. However, it is an acceptable limitation at the prototyping stage. In the future, when a new calculation method becomes available for non-domestic buildings, the prototype can be extended to cover it.

6.7.2 Average daylight factor

For the room daylight, CIBSE's average daylight factor model (CIBSE, 1987) will be used. In this model, the ratio of internal average illuminance (a spatial average over the working plane) to external global horizontal illuminance will be defined under the assumption of CIE overcast sky condition. It is especially useful at the early design stage, when designers needed to decide the approximate window area.

In this method the daylight factor for a room is determined by the window opening area, window frame and glazing type, internal surface areas and light reflectance properties.

6.7.3 Elemental U-value

Thermal transmittance, or U-value, of a building element represents its ability to conduct heat. The greater the U-value is, the more heat it can transfer through. Standard steady state U-value calculation is the model used (CIBSE Guide Section A3).

According to this model, the U-value of a building element depends on its construction, its surface conditions as well as external exposure conditions. Apart from the straight-forward calculation of elemental U-value, the operation can be reversed to calculate the thickness of a layer of the construction, say the insulation layer, in order to get a required U-value.

6.7.4 Compliance checking

For the purpose of energy conservation, the Building Regulations (for England and Wales) Part L sets standards to limit heat loss through the building fabric. There are two approaches to check the compliance of a building with the Building Regulations requirements (1990 edition). They are the *Elemental approach* and the *Calculation approach*.

Elemental approach:

This is a method to limit the U-value of the external building element and the window area of a building. The elemental U-value and window area requirement are given in the Building Regulations Approved Document. The Elemental approach for compliance checking is to calculate the U-value of each external element of the proposed building and its total window area then check against the standard. The building complies with the Building Regulations if the U-value of the external element and the window (including roof lights) area do not exceed those given in the Building Regulations.

Calculation approach:

The Calculation approach is an alternative to elemental approach. It allows variations in the level of insulation of individual elements and window area. There are two calculation procedures. One is to check the heat loss rate; the other is to check energy consumption of the whole building. In both procedures, the calculation result of the proposed building is compared with that of a notional building which has the same size and shape and complies with the requirements of the Regulation using the *Elemental approach*. If the former is not greater than the latter then the requirement is met. Otherwise the proposed building fails to comply with the Regulation.

6.7.5 Summertime temperature

If a southerly or westerly facing room has a large glazing area without any solar protection, overheating may occur during the summer time due to high solar gain. Overheating risk assessment is to calculate the peak temperature of a room then compare it with a standard, says, 27°C. If the summertime peak temperature does not exceed the standard, the thermal condition of the room is acceptable. Otherwise, it has overheating problem.

The calculation method suggested in CIBSE guide section A8 (1986) will be used in this prototype. According to this method, the summer time peak temperature depends on the climate condition of the location, window opening size and orientation, glazing type and shading facility, constructions of the external and internal room element, and so on.

6.8 Data and Process Analysis

Data and process are the essential components of a software program. During the software development process all the data entries and process modules have to be described explicitly and unambiguously.

Data analysis establishes the description of the design objects from the point of view of this prototype. It is a necessary step to establish the data model, or the aspect model in the COMBINE context, of MultiCAD.

Process analysis decomposes the BPE methods or tools into functional modules, which consolidates the data analysis and provides the basis for the software implementation.

The analysis task of the MultiCAD includes the following activities (table 6.3).

Table 6.3 Data and process analysis task

Activities	Objective	Output
Input & Output Data (Data analysis)	List out the input and output data of all the calculation methods. Compare for differences and commonalities.	A list of the input and output of all the calculation methods.
Data Documentation (Data analysis)	Document data using a unified format.	A dictionary of the data collection of MultiCAD.
NIAM Analysis (Data analysis)	Establish the relationship between sets of data. The first step towards establishing the data model.	A set of NIAM diagrams.
IDEF0 Analysis (Process analysis)	Associate data with functional modules. Leading to software design.	A set of IDEF0 diagram.

The first data analysis activity is to compile the input and output data of all calculation methods. The result is a list of data items expressed in more or less natural language.

In order to impose some degree of formalism to the way of describing data item, all the data items are documented using a unified format for the COMBINE project. The objective is to achieve semantic integration between different DTs, to guarantee a unified meaning of all items in the data dictionary.

In order to establish a data model or data schema, structured analysis is needed. There are many structured analysis techniques available, i.e., hierarchy charts, data flow diagrams, Warnier/ORR diagrams, NIAM, IDEF and so on. (Aktas, 1987). The choice of analysis technique for a particular data modelling task depends on the type of software used for the implementing applications based on the model. It is also, to some extent, a matter of taste (Björk, 1991a). During the analysis of MultiCAD, NIAM and IDEF0 are used.

In the following the strategies of data and process analysis for MultiCAD are discussed and examples are given to illustrate this work.

6.8.1 Primary I/O data

Data analysis starts with the building of the input and output data entries of each BPE model in the prototype. The work is carried out by examining the calculation methods and picking up the I/O data used in the processes. The following is a description of the analysis for daylight calculation.

The CIBSE daylight calculation equation is $DF = (TW\theta) / (A(1-R))$. Where T is the diffuse transmittance of glazing material including effects of dirt, W is the net glazed area of window (m^2), A is the total area of interior surface (m^2) (ceiling, floor, windows and walls), R is the area-weighted average reflectance of interior surfaces (ceiling and floor and walls, including windows), and θ is the angle in degrees subtended, in the vertical plane normal to the window, by sky visible from the centre of the window (CIBSE, 1987). The CIBSE window design guide specifies rules of deriving some data items above. For example, glazed area W equals to opening area minus window frame area which in turn depends on the window frame type. The area-weighted average light reflectance R is calculated by $R = (R_1 \times A_1 + R_2 \times A_2 + R_3 \times A_3 + \dots + R_n \times A_n) / (A_1 + A_2 + A_3 + \dots + A_n)$, A_1, A_2, A_3, A_n are areas of each internal surface and R_1, R_2, R_3 , and R_n are their corresponding light reflectance. In addition, CIBSE guide specifies a procedure for checking the adequacy of the daylight distribution in a room, which requires room length, width and window head height.

After all the data items are identified, they are associated with appropriated objects, i.e., building, room, element, etc. The derived data, such as area-weighted average light

reflectance R , is excluded because it can be calculated using other data. The objective is to identify the objects involved in the system and the attributes associated with them. The analysis result is used in the following data modelling exercise.

The same analysis is carried out for the other calculation methods, table 6.4 and table 6.5 shows the resulting data lists for all methods and a comparison between the data items.

Table 6.4 Input data comparison between different BPE methods

ITEM	BREDEM	COMPLIANCE	OVERHEATING	DAYLIGHT	U-VALUE
Site	<ul style="list-style-type: none"> - latitude, depends on site location. - monthly mean solar declination constant. - monthly average temperature, depends on site location. - monthly solar radiation on horizontal, depends on site location. 		<ul style="list-style-type: none"> - daily mean solar irradiance on window surface - daily peak solar irradiance on window surface - daily mean sol-air temperature - daily mean outdoor temperature 	<ul style="list-style-type: none"> - obstruction on windows 	<ul style="list-style-type: none"> - exposure condition
Building	<ul style="list-style-type: none"> (of site) domestic house only - building age - draught control level - zonal division - ventilation rate, depends on building age and draught control level. 	domestic building	(of site) any type of building	any type of building	any type of building
Zone	<ul style="list-style-type: none"> (of building) - name - net volume - floor area, measured between midpoint of the structure. - heating temperature - heating regime - proportion heated 				
Room			(of building) - volume - ventilation rate which depends on position and operation of windows	(of building) - width - length	

Element	(of zone) external walls, roofs, floors - name - area - U-value, depends on both element structure and surface exposure conditions.	(of building) exposed, semi-exposed and total floor - area - U-value, depends only upon the element structure.	(of room) external and internal elements - internal surface area - thermal admittance external element only - U-value, depends on both element structure and surface exposure - decrement factor - time lag	(of room) wall, floor, ceiling, door - internal surface area - surface reflectance	(of building) wall, roof, floor - structure type - layer material - layer thickness
Window	(of zone) - name - opening area - U-value - frame area - glazing transmission - dirtiness correction for heat gain. - orientation, measured from true north clockwise - external shading factor	(of building) - opening area - glazing type - U-value which is decided by glazing type	(of room) - glazing area - shading type - sun protection - usage pattern - mean solar factor, which is depends on the above three factors - U-value of glazing - thermal admittance of glazing - orientation, used to decide the surface solar irradiation	(of room) - opening area - frame area - glazing transmission - glazing reflectance - dirtiness correction for daylight - angle of visible sky - height of window head from floor surface	(of building) - opening area - frame type - glazing type
Occupants	(of building) - number		(of room) - number - time of duration		
Cooker	(of zone) - type				
Light & other casual heat source	(of building) as function of number of occupants and floor area		(of room) - instantaneous gains - time of duration		
Heating system	(of building) - type, with relevant parameter - efficiency				
Hot water system	(of building) - type for tank system - volume - insulation standard				

Table 6.5 Output data comparison between different BPE methods

ITEM	BREDEM	COMPLIANCE	OVERHEATING	DAYLIGHT	U-VALUE
Building	- annual and monthly energy consumption. - annually and monthly heat gain from solar, incidental and space heating.	- compliance result.			
Zone	- mean temperature. - annually and monthly energy consumption. - annually and monthly heat loss.				
Room			- peak temperature. - result of overheating assessment.	- average daylight factor. - result of daylight distribution checking.	
Element	- U-value - heat loss rate.	- U-value.	- surface reflectance.	- opening area.	- U-value - thickness of layer
Window	- opening area. - heat loss rate. - solar gain.	- opening area. - U-value.			- U-value

The tables indicate that the input and output are very different for these BPE methods. There are very few data items shared directly by more than one method although some might be described as being the same in natural language. For instance, elemental area is required by all the BPE methods except U-value calculation. However, the implication is very different in each case. In BREDEM building elements belong to a thermal zone and the area is measured from the mid-point of the inter-zone element to the internal surface of the external elements. In the daylight and overheating calculation, elements are considered as components of a room and the area is measured between the room internal surfaces. In the compliance checking, the element is referred to the building external elements and the area measurement is taken between the surfaces of the internal finishes of the external building elements.

The diversity of the I/O data of different BPE tools highlights the difficulty of data integration.

6.8.2 Unified data documentation

Following the initial input and output data analysis, the I/O data in MultiCAD is documented using a uniform format for the whole COMBINE project. The aim is to introduce certain degree of formalism into the data analysis process, because the initial I/O data analysis shows that misunderstanding and inconsistency is possible if natural language is used to document and communicate the data items.

The unified form used for the data documentation consists of three parts and seven sections (figure 6.7). The first part is information about the BPE program or method, the second part is about input data and third part is about output data.

```

START_FORM: IDCODE
1- Program or calculation method
  1.1- Name or identifier
  1.2- Version (if any)
  1.3- Date of release
  1.4- Contact name and address
  1.5- Brief description
  1.6- Main objects (entities)
  1.7- Further information
2- Purpose
  2.1- Definition
  2.2- Rules for interpretation of results (if any)
3- ID code for program-purpose
  3.1- IDCODE
START_DATA: IDCODE_I_nn
4- Object (related to input data)
  4.1- Name or identifier
  4.2- Brief description (if needed)
  4.3- Further information
5- Input data
  5.1- Name or identifier
  5.2- Description
  5.3- How used (purpose of use)
  5.4- How derived (method)
  5.5- Related objects (if any) or specific links to other data
  5.6- Type (format- real, integer, string, graph, etc.)
  5.7- Units
  5.8- Range
  5.9- Default value (if any)
  5.10- Source (of values, range etc.)
  5.11- Further information
START_DATA: IDCODE_O_mm
6- Object (related to output)
  6.1- Name or identifier
  6.2- Brief description (if needed)
  6.3- Further information
7- Output data
  7.1- Name or identifier
  7.2- Description
  7.3- Related objects (if any) or specific links to other data
  7.4- Type (format- real, integer, string, graph, etc.)
  7.5- Units
  7.6- Range
  7.7- Further information
END_FORM_IDCODE

```

Figure 6.7 Structure of the I/O data description form

In using this form, "sections 1 to 3 have to be given once for each Program-Purpose combination. Sections 4 and 5 have to be repeated for each and every input data item whereas sections 6 and 7 have to be repeated for each and every output data item" (Parand, 1990). An example of data documentation for Daylight calculation using this form is given in appendix A.

In the COMBINE project, the data documentation was carried out for every DT. The initial result has proven that each DT has its own view of the building and that the data integration

is a difficult task. For example, there are no less than seven different definitions for the term *building* from the six DTs (Dubois, 1991).

While this form is helpful in the clear definition of input and output data items, it is not a data modelling tool. It associates I/O data with objects, but does not support any relationship between different objects. There is no data modelling semantic knowledge underpinning this data documentation form. The data documented in such a form still needs to be manually interpreted for other purposes, such as establishing a data model.

6.8.3 NIAM analysis

NIAM is a structured data analysis and modelling technique. It presents the objects or data and their relationships in graphic form. Figure 6.8 is the conceptual building model for the daylight application expressed in NIAM. Diagrams for other BPE methods are given in Appendix B.

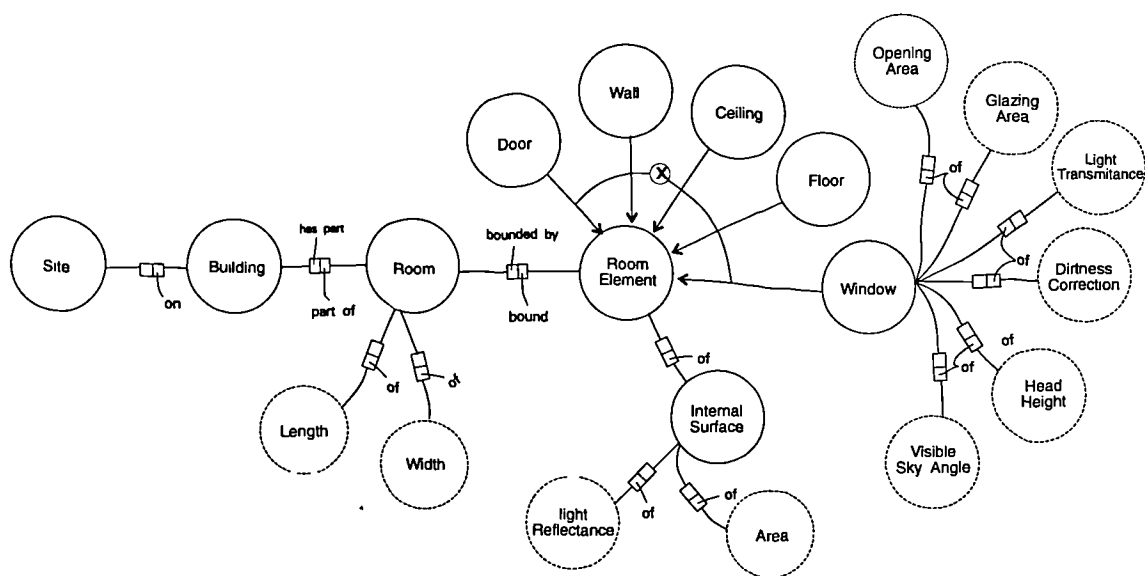


Figure 6.8 NIAM diagram of building model for daylight application

NIAM uses notations of *Non-Lexical* and *Lexical* objects which are represented by solid and dashed circles respectively in the graphics. Non-Lexical objects are usually real world entities such as *Building*, *Room* and so on; while Lexical objects are place holders for real values, i.e., room length and width. The relationships between objects are represented by role boxes which are shown as rectangle. The texts with the role box specify the nature of the relationship, for example a building "has" a room as a "part" of it; and a room is a "part of" a building. The arrow line between two objects represents "one of" or subtype relationship, i.e., a wall is a special room element. It inherits all the behaviour of a room

element and it also defines some specific behaviours. "X" in a cycle represents exclusive relationship, e.g., "door", "wall", "ceiling", "floor" and "window" are exclusive building elements. In other word, if a room element is a wall, by implication, it can not be a ceiling or a floor at the same time. It appears an unnecessary constraint in this example, it can be very important in some cases. For more information about NIAM technique, the reference (Turner 1989) could be consulted.

The NIAM diagrams clearly show the different conceptual building data schema for the five BPE tools. BREDEM decomposes the building into two heating zones. Compliance checking with the Building Regulation considers building as a structure of external building elements. Daylight and Overheating method break up the building into a collection of rooms. However, their views are still different, overheating concerns with the construction of the room element while daylight only needs the interior surfaces. U-value deals the cross section of individual elements. Figure 6.9 is a graphical illustration of the situation using a house as an example.

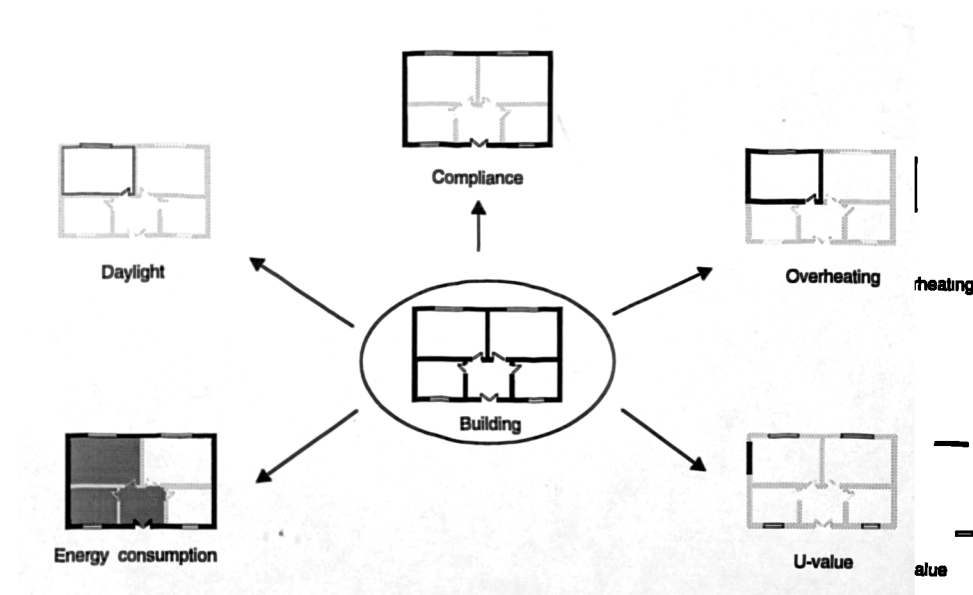


Figure 6.9 Different views of the building by different BPE tools

NIAM represents the data model graphically, it helps software developers to understand the problem they are dealing with. There are tools available, i.e., NESSIE (Kendall, 1992) and CGE (TNO, 1993), to translate NIAM model into ISO STEP/EXPRESS format which is evolving into an international standard data description language. This improves the efficiency of software development process, because the information produced at analysis stage can be used directly in the design and implementation phases.

However, there are some limitations of the NIAM modelling technique. It deals with static data but not dynamic behaviour or functions. For example, figure 6.8 shows a room has length, width and is bounded by many room elements. It does not show a room has a function for calculating average daylight factor, total internal surface area and so on.

In representing one-to-many relationship, NIAM does not distinguish whether the many members are exclusive or not exclusive, ordered or not ordered. These additional constraints can be significant in data modelling and software implementation. For instance, a building has many rooms, the order of a room member is not important. On the other hand, a polygon has many point coordinates, the order of presenting the points is important, because, different polygons can be generated if the order is changed. These issues are considered in the next stage data modelling of MultiCAD which is described in section 7.5 of chapter 7.

6.8.4 IDEF0 analysis

The objective of the process analysis is to identified the functional units of the BPE methods and the input/output data associated with them. This will have benefits in several respects: consolidating the DT's data analysis, providing the basis for software implementation and so on.

The process analysis of the prototype is carried out by using an IDEF0 technique whose structure is shown by figure 6.10 (Meta Software Corporation, 1987). The *Process name* in the box is the description of the process. *Input data* specifies all the input data required in the process, and *Output data* is the output of the process. *Control data* expresses the constraints on the process. The *Support mechanism* identifies the department or individual related to and/or responsible for the process.

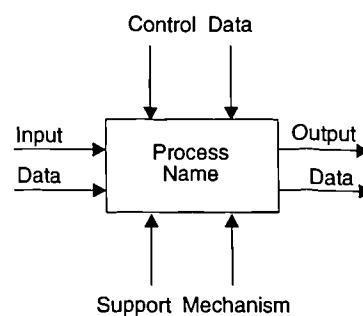


Figure 6.10 Basic structure of IDEF0

Figure 6.11, 6.12 are examples of the IDEF0 analysis for the daylight calculation. The analysis of a large system could result several hundred pages of IDEF0 diagrams. These diagrams are arranged in a hierarchy. In this example, 6.11 represents the top level, while

6.12 represents one level lower, in this case it is a further decomposition of the process A1 *determine window parameters*.

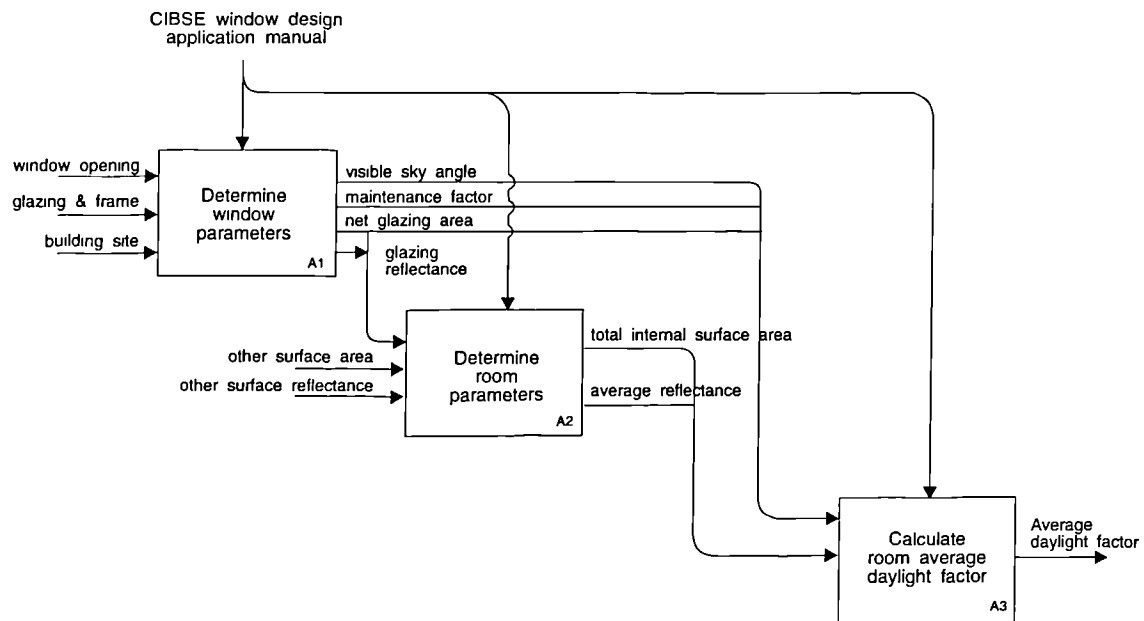


Figure 6.11 Average daylight factor

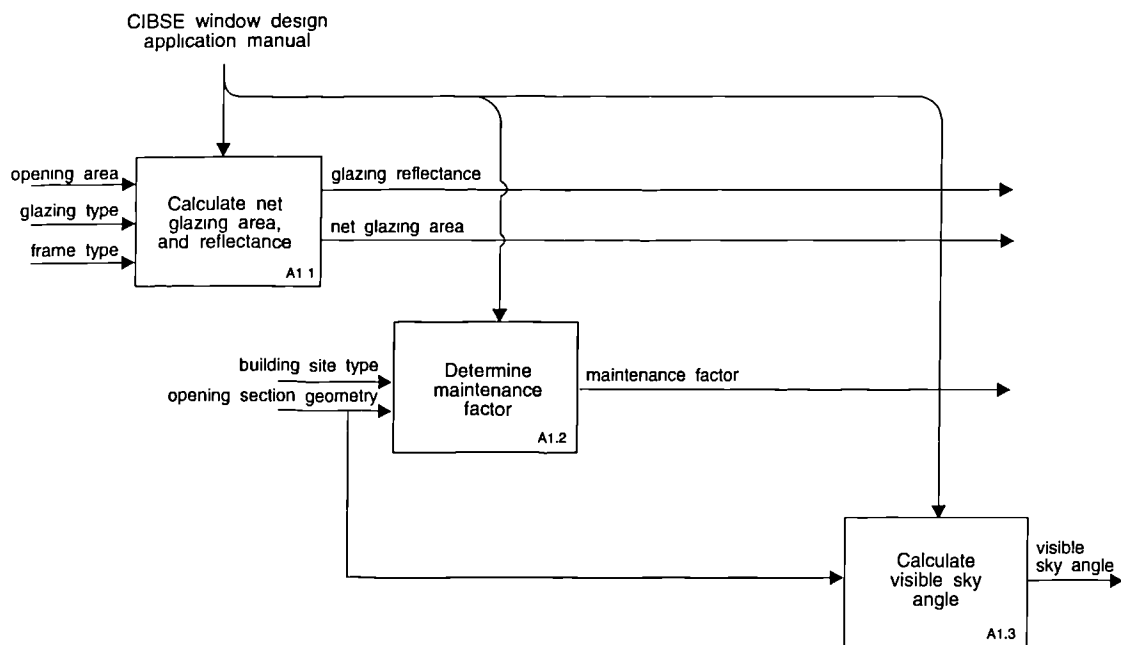


Figure 6.12 Determine window parameters

Figure 6.11 shows three processes in the daylight calculation for a room, A1 determine windows parameters, A2 determine room parameters and A3 calculate room average daylight factor. For each process the input data is represented by arrows and text on the left side and the output is on the right side of the box. For example, the process - determine windows parameters, takes window opening data, glazing and frame as well as building site information as input and produces output data for visible sky angle, maintenance factor, net glazing area and glazing reflectance. The figure also shows that output data of one process can be used as input data of another process, e.g., glazing reflectance is an output of process A1, but an input of process A2. The arrows and text on top of each box indicate that all the processes are carried out according to rules in the CIBSE window design application manual.

Figure 6.12 shows a further decomposition of process A1 which is also divided into three processes, A1.1 calculate net glazing area and reflectance, A1.2 determine maintenance factor, and A1.3 calculate visible sky angle. The syntax and meaning of the diagram are the same as that of figure 6.11.

Unlike NIAM, IDEF0 analysis covers dynamic behaviour as well as static data. However, in essence, it is an activity analysis technique and is structured around processes or functions of a software system. IDEF0 diagrams do not reveal the underlining data model, it is not used to define the data model for the specified activities or processes. Its major purpose is to identify the functional units and to associate data items with the functional units.

6.9 Summary

This chapter describes the first phase work of the prototype development. It defines the target users of the prototype, its design context. A design request analysis has been conducted, and on basis of the analysis, the system functional specification has been established. Five BPE-methods have been identified in order to support these functional requirements.

The data and process analyses identify the data entries of all the BPE methods in the prototype. This work provides the base for the establishment of the data model of the MultiCAD prototype, in the meantime, it contributes to the development of IDM of the COMBINE project.

NIAM and IDEF0 are two formal analysis techniques used during the analysis of MultiCAD. They are decompositional top-down approaches which break the system into software components, i.e., data entities, process modules and their inter-relationship. At the next stage

of system design, compositional bottom-up approach will be needed to re-assemble these components in a structured way.

CHAPTER 7

PROTOTYPE DEVELOPMENT, DESIGN AND IMPLEMENTATION

7.1 Introduction

Chapter 6 introduced the functional specification, data and process analysis of MultiCAD. This chapter describes its design and implementation. The major issues include:

- the establishment of the data model,
- the system structure of MultiCAD,
- the user interface design,
- the data mapping between MultiCAD and the COMBINE IDM,
- the coding process.

In addition, the software quality criteria is set out, and the targeted computer hardware and software environment for the implementation of MultiCAD are introduced.

7.2 Criterion of Software Quality

Meeting the functional requirements set out in the specification is obviously the primary target of the software design and implementation. In addition, there are some criteria for assessing the quality of a software product. Meyer (1988) identified five key software quality aspects as **correctness**, **robustness**, **extendibility**, **reusability** and **compatibility**. Another important quality aspect is the software **usability** which is highlighted in the discussion in section 5.3 of chapter 5 by the lack of usability of some of the existing software in architecture. The meaning and implication of the each of these quality aspects are defined as follows.

Correctness: Correctness is the ability of the software product to perform its tasks as defined by the requirements of its specification. First and foremost, a computer system should perform its tasks correctly, otherwise everything else about it matters little.

Robustness: Robustness is the ability of the software system to function even under unexpected conditions. The robustness of a system is an issue of what is happening outside the situation explicitly defined by the system specification. It involves the stability and reliability of the system. During the operation of a system, a user could take actions which are not anticipated during the development process. In these cases, the system should be able to act robustly in maintaining its integrity.

Extendibility: Extendibility is the ability with which the software product may be extended in response to specification changes. It is to be expected that, during the maintenance cycle, the task requirements for a system will change with time. When

this happens, a good system should be able to satisfy the new specification requirements with minimum changes.

Reusability: If a whole system or part of a system can be used in new system development, it will improve the efficiency of computer application development and reduce cost and the chance of errors.

Compatibility: It is the ability of the software to adopt to different hardware and operating system platforms. A software product would have a big advantage if it can operate on multiple hardware, i.e., PC, client server network, and operating systems, such as UNIX, DOS, MS-Windows, etc.

Usability: The ultimate objective of a software product is to be understood and used by its users, no matter how good it might be in other aspects. One important influencing factor in this respect is the user-system interface.

The above criteria will be used as guidance during the development of MultiCAD in order to achieve quality software. However, as a prototyping process, these criteria are given different priorities. The main objective of MultiCAD is to test the concept of an integrated building design system, therefore, the emphasis on software quality is put on its usability, extendibility and reusability. Correctness and robustness are also necessary requirement for the prototype in order to demonstrate the IBDS principle. However, the compatibility is not major concern at this stage. In this respect, MultiCAD is to be implemented for a computing platform that is widely used by architects in practice, porting to other platforms is an issue beyond the scope of this study.

7.3 Implementation Computing Environment

The computing environment has a very important impact on software design and implementation. Chapter 5 described recent developments in computer hardware and software. This section introduces the hardware and software environments specific to the implementation of MultiCAD.

7.3.1 Hardware environment

Surveys both in UK and other countries have shown that PCs have the greatest penetration in the building industry (Ray-Jones, 1990; Stevens, 1991). In most architectural design offices, one could expect to find a combination of workstation (minicomputers) and Personal Computers (PC, microcomputers). As a general trend, the unit cost of computing hardware is coming down while its performance is improving (section 5.2.1). At present, machines based on the 8086 or 80286 Central Processing Units (CPU) are already out of

date for any serious applications. The main stream of the PC market is machines with 80386 and 80486 CPUs. The targeted computing environment of MultiCAD is such a PC dominated one. The implementation of MultiCAD is for IBM PC 386, 486 and compatible equipment.

7.3.2 Software environment

One reason for the choice of the PC environment is that it has a wide range of high quality, relatively inexpensive, commercial software tools to aid the development, e.g., word-processing, graphic presentation, analysis, etc. These existing tools will be exploited to a maximum extent so that minimum effort is wasted on 're-inventing the wheel'. Specifically, the following software packages provide the platform for the prototype implementation.

MS-DOS	an industry standard operating system
Microsoft Windows	an industry standard graphical interface environment
Actor, from Whitewater Group	an object oriented programming environment
ObjectGraphics	an object oriented graphics library
WRT, from Whitewater Group	a tool for developing Windows application resources

In the following, these software packages in the implementation environment are briefly introduced.

MS-DOS and MS-Windows

MS-DOS provides a simple industry standard operating system, which is stable, mature, and has an easy upgrade path (Microsoft Corporation, 1991a). DOS 5.0 is used, which is the latest version available at the time of the prototype implementation. MS-Windows provides a high quality graphical user interface environment, with wide support because of its commercial success. The release of Windows 3.0 has solved a number of long-standing problems in memory management and multi-tasking, and allows the software developer access to advanced features and an existing base of software. The latest version 3.1 is a further enhancement of the windows environment (Microsoft Corporation, 1991b). The incoming Microsoft New Technology (NT) environment could provide an easier way for software migration to other computing platforms, such as UNIX (Udell, 1992). Microsoft Windows based applications represent a major spectrum of commercial software packages.

Actor

Object Oriented Programming (OOP) has become increasingly important during the past decade (Cox, 1991). C++, (an enhancement to the C language) is the main thrust in the OOP world. It can be used to produce efficient, compact and fast programs. However, the major disadvantage for the current generation of C++ compilers is that they are not very convenient for prototyping due to the torturous compile process. Fortunately, there are some other contenders in the OOP language world, which are easy to prototype but with compromises in program size and speed. For the PC environment, Estock (1989) reviewed three major object oriented packages, Smalltalk/V, Objective-C and Actor. He concluded that these three are all solid OOP systems. Each has desirable features for individual applications with Actor being particularly suited to the development of MS-Windows applications. This is a main consideration of this prototype, consequently, Actor is selected as the major programming language for the implementation of MultiCAD.

The Actor language has the following main features (Whitewater Group, 1991):

- Actor's code is natural language like and easy to understand
- Actor integrates fully with the MS-Windows environment, allowing real-time debugging and incremental development.
- There are more than 120 classes already available in Actor, which provide a wide range of functionality.
- Actor programs can be sealed off as 'executables' files which will run on any PC with MS-Windows.
- Actor comes with several accompanying packages, e.g., WRT and ObjectGraphics, for the purposes of building up resources and graphics.

Actor provides the environment for rapid programming in an object oriented fashion. The collection of the pre-defined Actor classes, methods as well as other accompanying application, are extremely useful. They allow effort to be concentrated on the real issue of solving the domain problems without being side-tracked into programming issues.

ObjectGraphics

Geometry and graphics are very important elements in the prototype. ObjectGraphics (Whitewater Group, 1990a) provides solutions to many problems in this respect. It is an extension of the Actor programming environment, and contains classes for general purpose of graphics, such as Shape, Brush, Pen, Line, Rectangle and so on. It can be used to generate a CAD like user interface which is able to generate and manipulate graphics.

Whitewater Resource Toolkit

MS-Windows applications usually have various graphical elements of user interface, such as dialog boxes, cursor, bitmaps and so on. These elements could be defined by application source code, or alternatively, they can be defined as separate resources. The advantage of separating resources, such as dialog box, menu, etc., from source code is that one can change the appearance of the application without accessing the source code. In addition, the resources can be used repeatedly.

Whitewater Resource Toolkit (WRT) (Whitewater Group, 1990b) is a package provided by the Whitewater Group. It is a powerful, yet easy to use tool for creating and managing the user interface of MS Windows applications. It includes a range of resource building tools for, i.e. bitmap, dialog, menu, icon and so on, which enables users to create or edit new resource in the MS-Windows environment. Moreover, it is designed to save resource directly into Windows executable files and eliminate the edit-compile-link cycle typically used in the software development.

Using WRT, all the resources are created through graphical interface instead of text. This enables an instant result inspection and no resource code writing is needed.

7.4 Object Oriented System Design

System design sets up the framework of transferring the system specification into programming code. In traditional software design, there are three distinct types of activities: external design, architectural design and detailed design (Fairley, 1985).

External design is a continuous activity starting from the requirement analysis. It is a process to specify the external recognisable characteristics of the software or its interface to the external environment. These characteristics include user display, report formats, performance requirement and so on.

Architectural design is to identify the modules in the software system and to associate data with the modules. It specifies the conceptual structure of the software.

Detailed design further specifies the module and data structure in the system. Details of the implementation are worked out at this stage.

Between the two essential facets of a software system, data and process, the norm of the software design in the traditional procedural programming is to focus on the process aspect of the system. The design task is to define the functional modules and to associate input and output data with them. Subsequent functional changes could lead to a complete re-design of the system. The potential instability of software functionality in the longer term

underlines one of the major deficiencies of the traditional procedure or process oriented system design.

The object oriented approach in software design and implementation takes a different view. It centres the system around data or object instead of process or function. In this part of the discussion on software design and implementation, data and object are two terms used interchangeably, so are function and process.

Meyer (1988) defined object-oriented design as being "the construction of software systems as structured collections of abstract data type implementations". The concepts of the object oriented paradigm are well documented in many sources (Meyer 1988, Whitewater Group, 1991, Kim 1990b). The following is a summary of the key terminology:

- **Object.** An object is a basic building block of an object-oriented program. It incorporates both data and functions in one functional unit.
- **Class.** A class is a term used to describe a set of objects of the same type. It is a template for creating objects. The class defines the methods and attributes shared by the set of objects.
- **Instance variable.** Instance variables are the data items of an object.
- **Method.** Methods define the function or behaviour of an object.
- **Message.** Message is the media of communication between objects.

An object oriented system consists of objects and the same type of objects are defined by a class. The tasks of object oriented system design is to identify the object classes and specify the relationship between the object classes.

There are advantages in the key aspects of the software engineering in structuring the system around data or object, for example, improved modularity, code efficiency, consistency and so on:

Modularity: Almost all software is made of smaller code units structured in certain way. These units or modules are also known as subroutines in traditional computing terms. In a good system, modules are autonomous and coherent. The object oriented approach produces good system modularity, because its basic elements - objects are modules in their own right.

Inheritance: Inheritance is the technique for bringing generic code into play when producing new code. In an object oriented paradigm, the essential elements of the system are object classes. Classes can be considered as module templates that can perform certain tasks on its data. Without inheritance, every new class has to define the services it offers. This will undoubtedly result in repetition of coding. Inheritance provides a solution to this

problem. It enables descendants to inherit all the attributes and services of their ancestors. The descendants only need to define the behaviour specific to themselves. For example, building "Element" class specifies all the properties and behaviours common to all the building element, such as geometry and surface area. A window opening is a special type of building element. It has a glazing and a window frame. In this case, a "Window Opening" can be defined as a descendant of building "Element". For the new "Window Opening" class, it is only necessary to define the attributes and operations which are special to window openings such as glazing area, frame area and so on. The geometry and surface area operation are inherited from its ancestor, the "Element". This feature of object oriented programming greatly improves software efficiency, reusability and extendibility.

Extendibility and reusability: Systems developed using the object oriented paradigm are easily extendible due to their well-modular nature. Any changes or additions to the system specification usually only require a relatively straightforward increase of new modules to the system rather than fundamental changes to the system architecture.

Another valuable merit of an object oriented system is its reusability. Most of the object oriented programming environments themselves contain large amounts of source code that can be used in new system development. Similarly, any code produced during the process of system implementation can be used in future development.

Polymorphism and encapsulation: These two concepts are at the heart of the power of the object oriented approach of software development. Polymorphism implies using the same message to communicate with different type of objects. This simplifies the programming greatly, since when sending a message to an object, the message only specifies what to do not how to do it. The object itself interprets the message by applying the built-in methods. Encapsulation means information hiding (Graham, 1992). In other words, each object keeps its internal structure to itself, other objects are not allowed to access the internal structure directly. The only way of data interrogation is through the methods defined by objects. This feature increases the control by the developer on the software and improves the security of the software.

7.5 Data Model of MultiCAD

A study of data modelling at University of California has identified three generations of database system (DBMS committee, 1991). The first generation is represented by the network and hierarchical data model and database system. The second generation is the entity-relationship data model and relational database. Recently, object oriented (OO) data

model and database systems have been emerging, they are considered as third generation databases.

The advantages of the OO data modelling have been argued by a number of authors (Meyer, 1988; Kim, 1990b; Graham, 1992). They can be summarised as follows:

- OO data models can better represent the reality, because it can be argued that the real world existence is either real object, i.e., building, wall, window, or abstract object such as geometry and topology.
- OO data models are better equipped for long term system evolution. When a change takes place to the software function aspect, the objects involved do not normally change. If this happens, a software system built upon an OO data model only needs to modify some of the behaviour of some of its objects.
- It is more efficient to organise software system by using object inheritance. Common attributes and behaviours can be defined by ancestors and shared by all the descendants.

In an object oriented paradigm, there are two aspects associated with an object, its attributes and its methods. The attributes define the static state of the object, for example a wall has a name, a construction, etc. The methods define the dynamic behaviour and functions of the object, e.g., a wall has a method of U-value calculation. The task of data modelling is to identify the object classes from the real world context, to specify their attributes and behaviour as well as to establish the relationship between the classes.

This section concentrates on the static aspect of the data model and the dynamic aspect is introduced in section 7.9. Two steps are required to produce the static data model of MultiCAD; firstly the identification and specification of the object classes in the model; and secondly the establishment of the relationship between different classes.

The data modelling described in this section is built on the basis of the data analysis introduced in Chapter 6. Most of the object classes identified during the data analysis are included in the data model, they are Site, Climate, Building, BredemZone, Room, Element, Wall, Floor, GloundFloor, Roof, Opening, WindowOpening, Window, Construction, Layer, TransluentLayer, Material, Finish, Equipment and Cooker. In addition several object classes are added into the model for the purpose of building geometry and topology, they are Network, BGMNetwork, NetArc, NetNode and 3DPoint. The idea behind the geometry and topology modelling is explained in section 7.5.1.

One type of relationship comes out of the NIAM data analysis is specialisation or 'is a' relationship. For example, Wall is a specialisation of Element, so are Floor, Roof and Opening. In this example, Element is the ancestor; Wall, Floor, Roof and Opening are

descendants. A descendant can, in turn, be an ancestor of other objects, e.g., Floor is an ancestor of GroundFloor and Opening is an ancestor of WindowOpening. There are two special cases in the specialisation hierarchy of the data model. The Room is considered as a general space class. Site and BredemZone are modelled as special kind of spaces, thus they descend from Room. This arrangement is very useful for the topology modelling. The value of a netnode is an instance of Room. Because Site and BredemZone are descendants of Room, instances of these two classes can also be netnodes in the topology model. Further details are given in section 7.5.1.

One important type of relationship, in the data model, is aggregation or 'is part of' relationship. For example, Room is part of Building, Layer is part of Construction, etc. Another type of relationship is the association relationship between objects. In these cases, one object often needs to reference another object, however, the object being referenced is not part of the first object. For example, Building has an attribute Site, but Site is not part of Building. In real world context, the association relationship includes relations such as 'is connected to', 'is used by', 'located at', and so on.

Figure 7.1 shows the data model of MultiCAD.

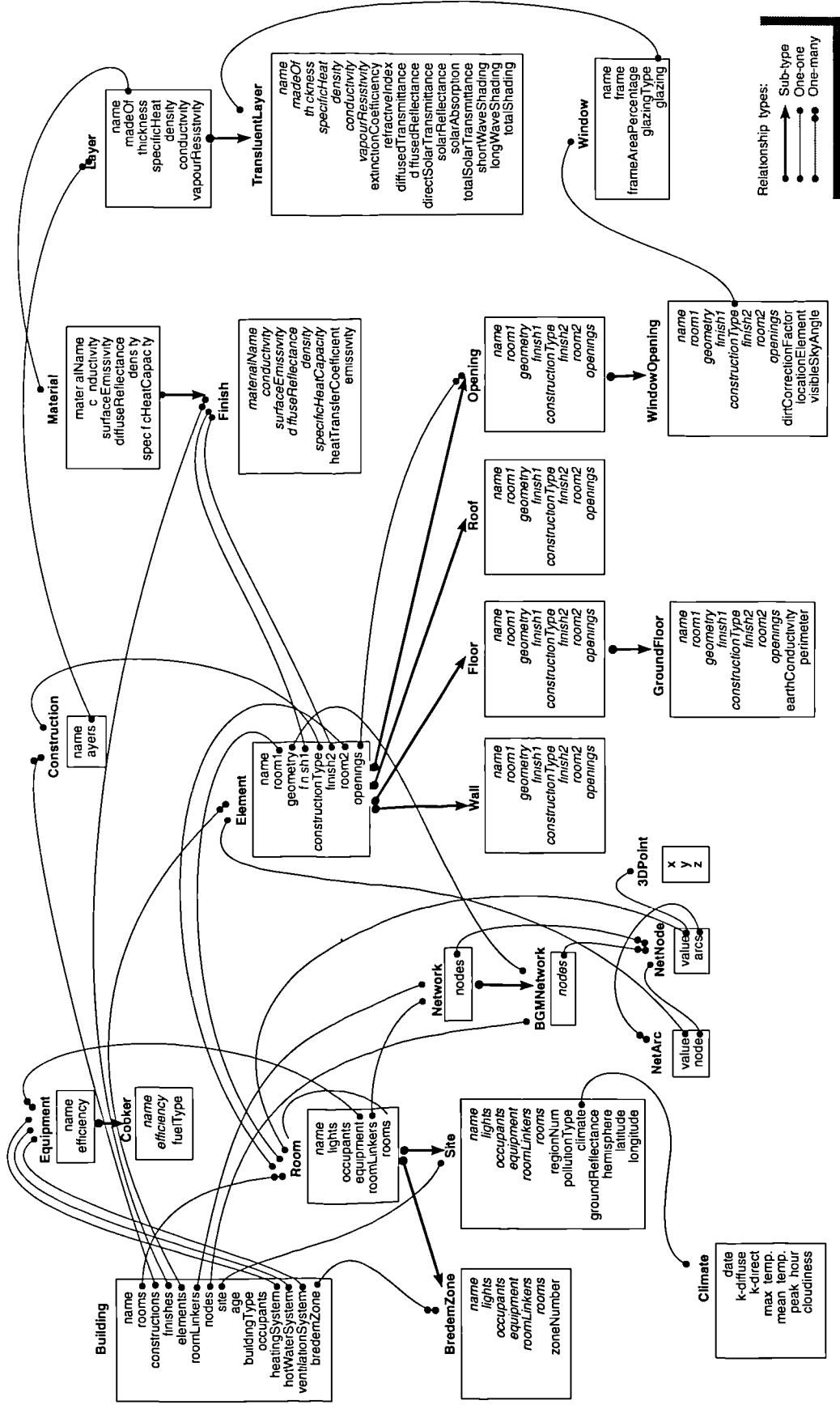


Figure 7.1 Data model for MultiCAD

The class name is shown on top of each rectangle in figure 7.1. The items in the rectangles represent the instance variables of the class. Italic items represent variables inherited from the ancestor class(es). Arrow line links show the subtype or descendent relationship, i.e., **Wall** and **Opening** are descendants of **Element**. Other lines represent aggregation and association relationships. A line link which has one dot on each end shows the 'one to one' relationship between the instance variable and an object of another or the same class. For example **Building** has a variable *site* which is an object **Site**. Lines with one dot on one end and two dots on the other represent 'one to many' relationships between a variable and objects of other or the same class. For example, an object of class **Construction** could contain several layers. This relationship is shown in the model. In the 'one to many' relationship, the many objects are sometimes simple collection, and sometimes ordered collection. The distinction is not shown in figure 7.1, it is explained in the following complete documentation of the building data model.

Building

name	(a String)
rooms	(a set of Room)
constructions	(a set of Construction)
finishes	(a set of Finish)
elements	(a set of Element)
roomLinkers	(a Network)
nodes	(a BGMNetwork)
site	(a Site)
age	(an Int)
buildingType	(a String)
occupants	(an Int)
HeatingSystem	(an Equipment)
hotWaterSystem	(an Equipment)
ventilationSystem	(an Equipment)
bredeZone	(a BredeZone)

Room

name	(a String)
lights	(a Real)
occupants	(a Int)
equipment	(a set of Equipment)
roomlinkers	(a Network)
rooms	(a set of Room)

BredeZone (descendant of Room)

<i>name</i>	(a String)
<i>lights</i>	(nil)
<i>occupants</i>	(nil)
<i>equipment</i>	(nil)
<i>roomLinkers</i>	(a Network)
<i>rooms</i>	(nil)
zoneNumber	(an Int)

Site (descendant of Room)

<i>name</i>	<i>(a String)</i>
<i>lights</i>	<i>(nil)</i>
<i>occupants</i>	<i>(nil)</i>
<i>equipment</i>	<i>(nil)</i>
<i>roomLinkers</i>	<i>(a Network)</i>
<i>rooms</i>	<i>(nil)</i>
<i>regionNum</i>	<i>(an Int)</i>
<i>pollutionType</i>	<i>(a String)</i>
<i>climate</i>	<i>(a Climate)</i>
<i>groundReflectance</i>	<i>(a Real)</i>
<i>hemisphere</i>	<i>(a String)</i>
<i>latitude</i>	<i>(a Real)</i>
<i>longitude</i>	<i>(a Real)</i>

Element

<i>name</i>	<i>(a String)</i>
<i>room 1</i>	<i>(a Room)</i>
<i>geomatry</i>	<i>(a BGMNetwork)</i>
<i>finish1</i>	<i>(a Finish)</i>
<i>constructionType</i>	<i>(a Construction)</i>
<i>finish2</i>	<i>(a Finish)</i>
<i>room2</i>	<i>(a Room)</i>
<i>openings</i>	<i>(an orderedCollection of Opening)</i>

Wall (descendant of Element)

<i>name</i>	<i>(a String)</i>
<i>room1</i>	<i>(a Room)</i>
<i>geometry</i>	<i>(a BGMNetwork)</i>
<i>finish1</i>	<i>(a Finish)</i>
<i>constructionType</i>	<i>(a Construction)</i>
<i>finish2</i>	<i>(a Finish)</i>
<i>room2</i>	<i>(a Room)</i>
<i>openings</i>	<i>(an orderedCollection of Opening)</i>

Roof (descendant of Element)

<i>name</i>	<i>(a String)</i>
<i>room1</i>	<i>(a Room)</i>
<i>geomatry</i>	<i>(a BGMNetwork)</i>
<i>finish1</i>	<i>(a Finish)</i>
<i>constructionType</i>	<i>(a Construction)</i>
<i>finish2</i>	<i>(a Finish)</i>
<i>room2</i>	<i>(a Room)</i>
<i>openings</i>	<i>(an orderedCollection of Opening)</i>

Floor (descendant of Element)

<i>name</i>	<i>(a String)</i>
<i>room1</i>	<i>(a Room)</i>
<i>geomatry</i>	<i>(a BGMNetwork)</i>
<i>finish1</i>	<i>(a Finish)</i>
<i>constructionType</i>	<i>(a Construction)</i>
<i>finish2</i>	<i>(a Finish)</i>

<i>room2</i>	<i>(a Room)</i>
<i>openings</i>	<i>(an orderedCollection of Opening)</i>

GroundFloor (descendant of Floor)

<i>name</i>	<i>(a String)</i>
<i>room1</i>	<i>(a Room)</i>
<i>geomatry</i>	<i>(a BGMNetwork)</i>
<i>finish1</i>	<i>(a Finish)</i>
<i>constructionType</i>	<i>(a Construction)</i>
<i>finish2</i>	<i>(a Finish)</i>
<i>room2</i>	<i>(a Room)</i>
<i>openings</i>	<i>(an orderedCollection of Opening)</i>
<i>earthConductivity</i>	<i>(a Real)</i>
<i>perimeter</i>	<i>(a Real)</i>

Opening (descendant of Element)

<i>name</i>	<i>(a String)</i>
<i>room1</i>	<i>(a Room)</i>
<i>geomatry</i>	<i>(a BGMNetwork)</i>
<i>finish1</i>	<i>(a Finish)</i>
<i>constructionType</i>	<i>(a Construction)</i>
<i>finish2</i>	<i>(a Finish)</i>
<i>room2</i>	<i>(a Room)</i>
<i>openings</i>	<i>(an orderedCollection of Opening)</i>

WindowOpening (descendant of Opening)

<i>name</i>	<i>(a String)</i>
<i>room1</i>	<i>(a Room)</i>
<i>geometry</i>	<i>(a BGMNetwork)</i>
<i>finish1</i>	<i>(a Finish)</i>
<i>constructionType</i>	<i>(a Construction)</i>
<i>finish2</i>	<i>(a Finish)</i>
<i>room2</i>	<i>(a Room)</i>
<i>openings</i>	<i>(a orderedCollection of Opening)</i>
<i>dirtCorrectionFactor</i>	<i>(a Real)</i>
<i>locationElement</i>	<i>(an Element)</i>
<i>visibleSkyAngle</i>	<i>(a Real)</i>
<i>altSolarGainFactor</i>	<i>(a Real)</i>
<i>solarGainFactor</i>	<i>(a Real)</i>

Window

<i>name</i>	<i>(a String)</i>
<i>frame</i>	<i>(a String)</i>
<i>frameAreaPercentage</i>	<i>(a Real)</i>
<i>glazingType</i>	<i>(a String)</i>
<i>glazing</i>	<i>(a TranslucenttLayer)</i>

Construction

<i>name</i>	<i>(a String)</i>
<i>layers</i>	<i>(a orderedCollection of Layer)</i>

Layer

<i>name</i>	<i>(a String)</i>
-------------	-------------------

madeOf	(a Material)
thickness	(a Real)
specificHeat	(a Real)
density	(a Real)
conductivity	(a Real)
vapourResistivity	(a Real)

TranslucentLayer (descendant of Layer)

<i>name</i>	<i>(a String)</i>
<i>madeOf</i>	<i>(a Material)</i>
<i>thickness</i>	<i>(a Real)</i>
<i>specificHeat</i>	<i>(a Real)</i>
<i>density</i>	<i>(a Real)</i>
<i>conductivity</i>	<i>(a Real)</i>
<i>vapourResistivity</i>	<i>(a Real)</i>
extinctionCoefficient	(a Real)
refractiveIndex	(a Real)
diffusedTransmittance	(a Real)
diffusedReflectance	(a Real)
directSolarTransmittance	(a Real)
solarReflectance	(a Real)
solarAbsorption	(a Real)
totalSolarTransmittance	(a Real)
shortWaveShading	(a Real)
longWaveShading	(a Real)
totalShading	(a Real)

Material

materialName	(a String)
conductivity	(a Real)
surfaceEmissivity	(a Real)
diffuseReflectance	(a Real)
density	(a Real)
specificHeatCapacity	(a Real)

Finish (descendant of Material)

<i>materialName</i>	<i>(a String)</i>
<i>conductivity</i>	<i>(a Real)</i>
<i>surfaceEmissivity</i>	<i>(a Real)</i>
<i>diffuseReflectance</i>	<i>(a Real)</i>
<i>density</i>	<i>(a Real)</i>
<i>specificHeatCapacity</i>	<i>(a Real)</i>
heatTransferCoefficient	(a Real)
emissivity	(a Real)

Equipment

name	(a String)
efficiency	(a Real)

Cooker (descendant of Equipment)

<i>name</i>	<i>(a String)</i>
<i>efficiency</i>	<i>(a Real)</i>
fuelType	(a String)

Climate

date	(a Date)
k-diffuse	(a Real)
k-direct	(a Real)
maxTemperature	(a Real)
meanTemperature	(a Real)
peakHour	(a Int)
cloudiness	(a Real)

Network

nodes	(a set of NetNode)
-------	--------------------

BGMNetwork (descendant of Network)

<i>nodes</i>	<i>(a set of NetNode)</i>
--------------	---------------------------

NetNode

value	(a Room or Point3D)
arcs	(an orderedCollection of NetArc)

NetArc

value	(an Element)
node	(a NetNode)

Point3D

x	(a Real)
y	(a Real)
z	(a Real)

The above documentation shows the class name in bold text, i.e., **Building**, **Room**, **Site** etc. The inheritance relationship is included, e.g., Wall is a descendant of Element. The attributes of each class are listed below the class name. The value type of each attribute is indicated after the attribute name in bracket. For example, the name of Building is a string, the constructionType of Element is a Construction.

The attribute values can be classified into four groups, primitive type, user defined object type, collection of user defined object type and nil. Primitive types, including String, Real, Int, are basic data types common to many computer programming language, they are the essential components based on which other data structures are constructed. When an attribute is a user defined object type, it usually represents a relationship between two objects, for instance, the site of Building is a Site and finish1 of Element is a Finish. In some cases, the value of an attribute is a collection of other types of objects; rooms of Building is a set of Room; layers of a Construction is an OrderedCollection of Layer.

The major difference between set and OrderedCollection is that in the first case the order index in the collection is not important while it is in the later case. For example, the sequence of presenting the rooms in a building does not represent any significance, we can

say a building has four rooms, a bedroom, a living room, a toilet and a kitchen or it has four rooms, a living room, a kitchen, a toilet and a bedroom. However, in other cases the sequence is important, change of sequence can result in a different object. A construction with layers of external leaf, insulation, cavity and internal leaf is different from one with layers of external leaf, cavity, insulation and internal leaf. Another difference between set and OrderedCollection is that set is an exclusive list, one object cannot appear more than once in a set while OrderedCollection is not exclusive, an object can appear more than once.

Another feature which needs to be pointed out is that although some descendants look identical to their ancestor in terms of the attribute list, they are different in behaviour aspect. This is the reason for making them descendants. For example, Wall and Floor are both descendants of Element, they have the same attributes as Element. However, they have a different way of calculating their U-value.

7.5.1 Features of the data model

The data model presented above is a conceptual model which can be implemented in any OO environment. The following issues emerged as important during the establishment of this data model, object nesting, geometrical modelling, object ownership and data consistency.

Object nesting: Most of the object classes in this model are complex classes which have instance variables of other classes. For example, an element has a constructionType which is an instance of construction and possibly a window opening of opening class. The operand of an object having another object as its instance variable is also known as object nesting. Figure 7.2 is an example of the object nesting which shows an element has a construction and an opening which in turn has a window as its construction. This type of object nesting can extend to many levels.

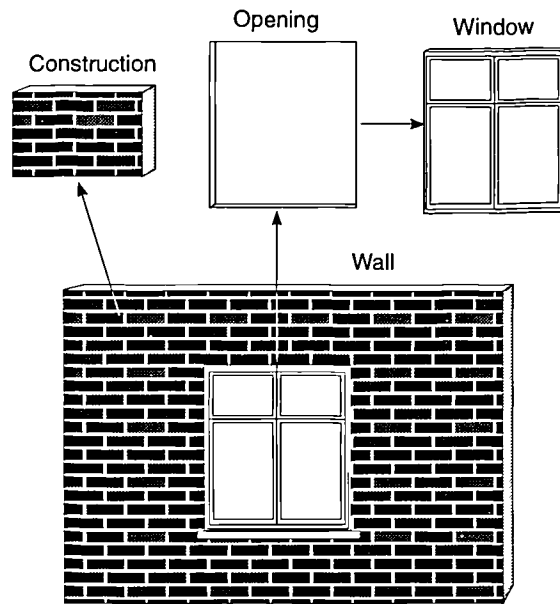


Figure 7.2 An example of object nesting

The data model of MultiCAD allows recursive object nesting. An object can have objects of its own class as instance variables. For example, the room class has an instance variable item - rooms, which is a collection of room objects.

Geometry and topology modelling: Another research project in the Department, the Energy Kernel System (EKS), has provided important input to the geometry and topology modelling of MultiCAD (Wright, 1992).

A building can be considered as two types of system, a structural system and a spacing system. The structuring system is about the shape of the building element such as walls, floors, roofs and so on. The spacing system is about the spaces formed by the building elements, i.e., rooms, zone and site. In this prototype, site is modelled as a special space. The objective of geometry and topology modelling is to provide a framework for storing geometry and topology information about the building and retrieving the information when needed. The geometry is about the dimensional information of the building and its components, i.e., the coordinates of the four corners of a wall; and topology is about the positional relationship between the building components, i.e., a room is connected to several particular walls.

MultiCAD adopts two types of network for the geometry and topology modelling, wire frame network and roomlinker network (figure 7.3).

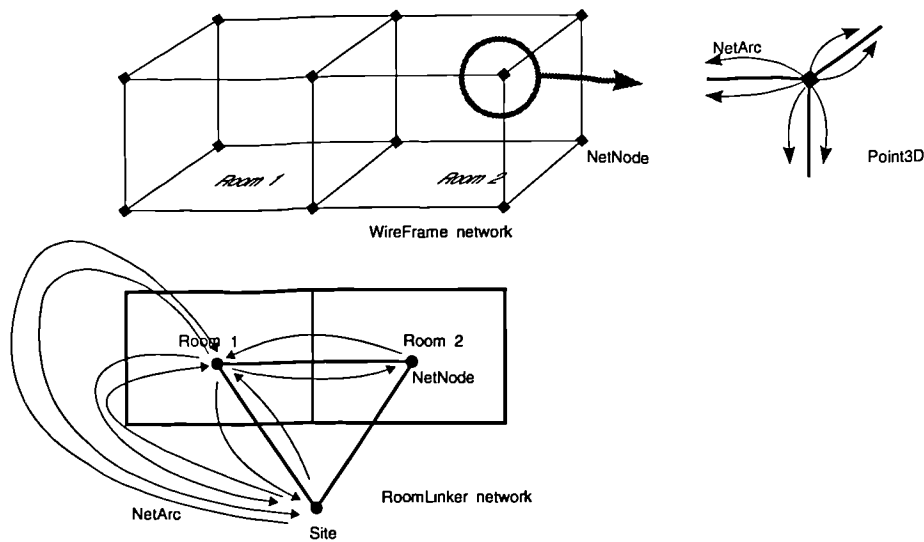


Figure 7.3 Geometrical and topological modelling

The wire frame network models the building structure and the roomlinker network models the building spacing system. These two networks are not independent of each other but rather two aspects of a large model. They both use the concepts of NetNode and NetArc. In the wire frame network, net nodes are 3D points which are joints of the building structure; net arcs are elements which form the link between joints. In the roomlinker network, net nodes are rooms or spaces and net arcs are boundaries between spaces.

Object ownership: There is a logical object ownership in the MultiCAD data model, which essentially reflects the real world relationship between objects. The significance of determining the object ownership is to specify the right of changing and deleting objects. This following is an example to illustrate the issues involved.

In a real building design, an architect may use one type of finish on many building elements. During the design, he may decide to make some changes on the finish of a particular building element, such as a wall. When he does that, he does not want changes to other elements which use that original finish. On the other hand, it is possible that the architect sometimes decides to change a particular finish on all elements in the building to another one. In this case, the architect does not want go through all the elements to change their finish, he just wants to say 'change finish A to finish B in the building'.

In the MultiCAD data model, an element object has instance variable *finish1* and *finish2* to represent its finishes on both sides of the construction. However, these two finish objects do not belong to the element object but to the building object. All the element has are pointers or references to the finish objects. The implication is that changes cannot be

introduced to the particular finish objects from the element, since other elements could have pointers to the same finish. It is possible to edit the finish variables of an element by creating new finish objects. Similarly, construction objects are not owned by elements, nor are material objects owned by constructions.

Data consistency: One measure to ensure data consistency in the model is to differentiate object data from object behaviour. Object data is the state of the object; it tells what the object is or what it has. Object behaviour is about how the object behaves in certain respects. Object data is stored as instance variables and object behaviours are defined by methods. For example, an element has a construction type as an instance variable. On the other hand, U-value is not an instance variable but a behaviour of an element, because U-value is a measure of the thermal behaviour of an element depending on its construction and surface conditions. Therefore, the U-value of an element is not stored as data but calculated using related methods. The same principle applies to elemental area, room volume and so on.

While the author is responsible for the initial data and process analysis, the establishment of the data model is a collaborative work with several colleagues (Lockley, Sun, 1992).

7.6 System Structure of MultiCAD

Architectural design is a multi-tasking, interactive, and designer control process as concluded in section 3.3.4. These characteristics of architectural design underline some fundamental requirements for the computer based design support systems, which include:

- Supporting multiple tasks.
- Interactive and integrated operation.
- Responsive, flexible and under the users' control.
- Extendible to include more functions.

MultiCAD tries to satisfy these demands. Figure 7.4 presents the conceptual system structure of the prototype. It is not the architectural structure of the system, since in an object oriented paradigm the software system is structured around data rather than functional modules.

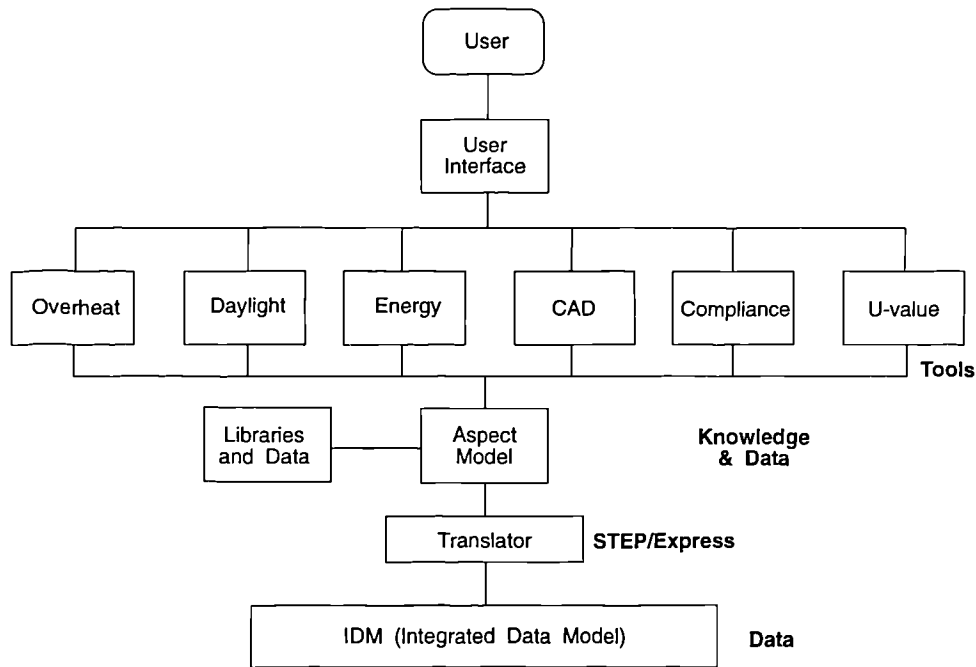


Figure 7.4 System configuration of MultiCAD

Each of the elements is explained as follows.

User Interface is the external appearance and the communication media of the prototype. It will be a computer screen with multiple windows driven by mouse and keyboard, making use of the given operating system style of windows and conforming to common windows application standards.

Tools are the functions of the prototype including Overheating, Daylight, Building energy, Compliance, Element Heat Loss and so on. These tools will be integrated and able to run concurrently and maintain data exchange dynamically. For example, the user may change a wall construction in the U value calculation and the change will be reflected in other related tools.

Aspect model contains the data information for the building under consideration. It is an implementation of the data model described in section 7.5. It will communicate with the COMBINE Integrated Data Model (IDM) for the purpose of data exchange.

Libraries and Data will be databases holding information such as material properties, manufacture products, climatic data and so on.

The **translator** provides the data translation between the aspect model and the IDM. STEP/Express protocol is used for the data exchange.

The **IDM** is the central data model of the COMBINE. It will provide the data exchange between different DT prototypes.

Data integration is the major concern of the COMBINE prototypes. Figure 7.5 shows the integration structure of MultiCAD.

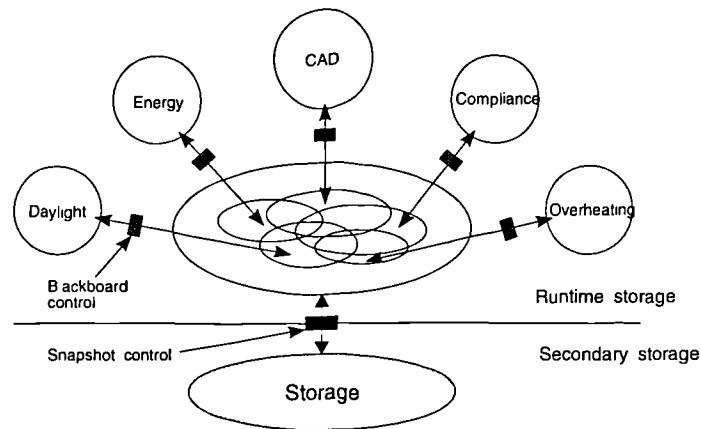


Figure 7.5 Integration structure of MultiCAD

The CAD application is used as a front end interface for the creation and the display of the building object. When a building is created or loaded from storage, the information concerning the building is stored in runtime memory in the frame of the OO data model. Any of the application tools can access to the data and perform evaluations on it. **Blackboard control** is a control mechanism which provides an optional temporary break between an application tool and the data model. It allows the user to concentrate on a particular evaluation without triggering off other unwanted operations. Any data changes by a tool can be stored in a buffer area in memory. A further confirmation of accepting or abandoning the changes can be made at the blackboard control point before putting the data back into the data model. Similarly the results of the operating session could be put back permanently to the secondary storage or be abandoned at the **Snapshot control** as shown in figure 7.5.

Although the tools are considered as functional modules conceptually, they are actually methods around objects or data (figure 7.6). For example, daylight, overheating are functions (methods) of room objects; energy is a method of a building object, etc. The invoking of these functions is done by sending messages and calling the methods in appropriate objects. This principle will be reflected in the interface design of this prototype.

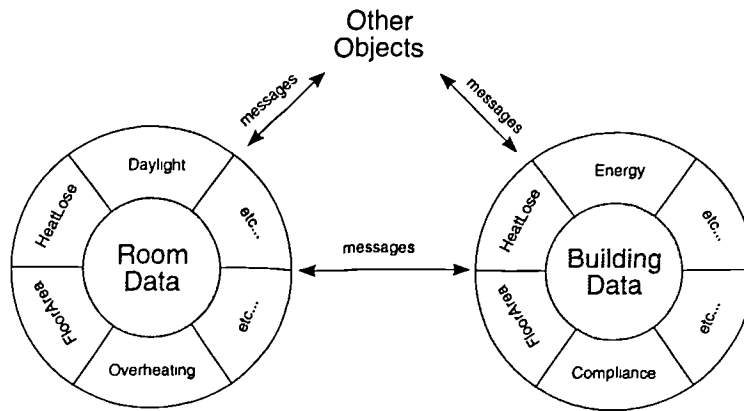


Figure 7.6 Data and methods in MultiCAD

7.7 User Interface Design

Discussions in Chapter 4 revealed that the user interface is often a common deficiency in many existing systems, and recent software developments have put significant emphasis on the user interface design to improve usability. MultiCAD system takes full advantages of user interface resources of the Microsoft Windows environment. Figure 7.7 shows the style of the user interface of the prototype.

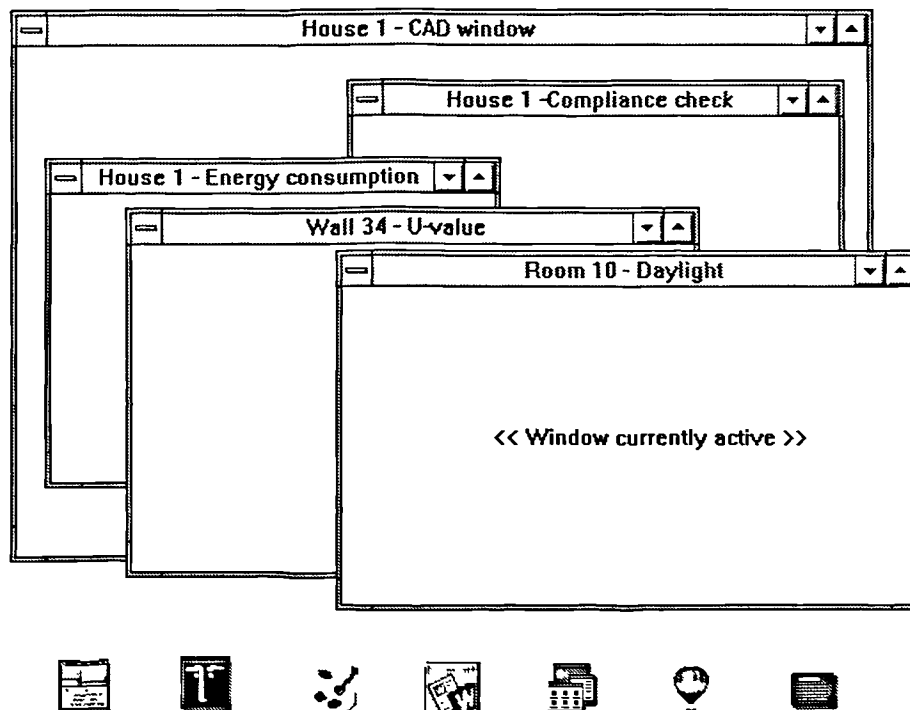


Figure 7.7 User interface style

The user interface of MultiCAD is not organised centred on the functional modules, i.e., energy, daylight, but centred on the objects, i.e., building, room, wall, etc. An editor is designed for the evaluation and editing of each object and each editor behaves as a tool on the designer's desktop. Functional applications are organised within the appropriate editor window. For example, the building energy application is put under building editor; room daylight application is put under room editor and so on. The interface structure for MultiCAD is shown in figure 7.8.

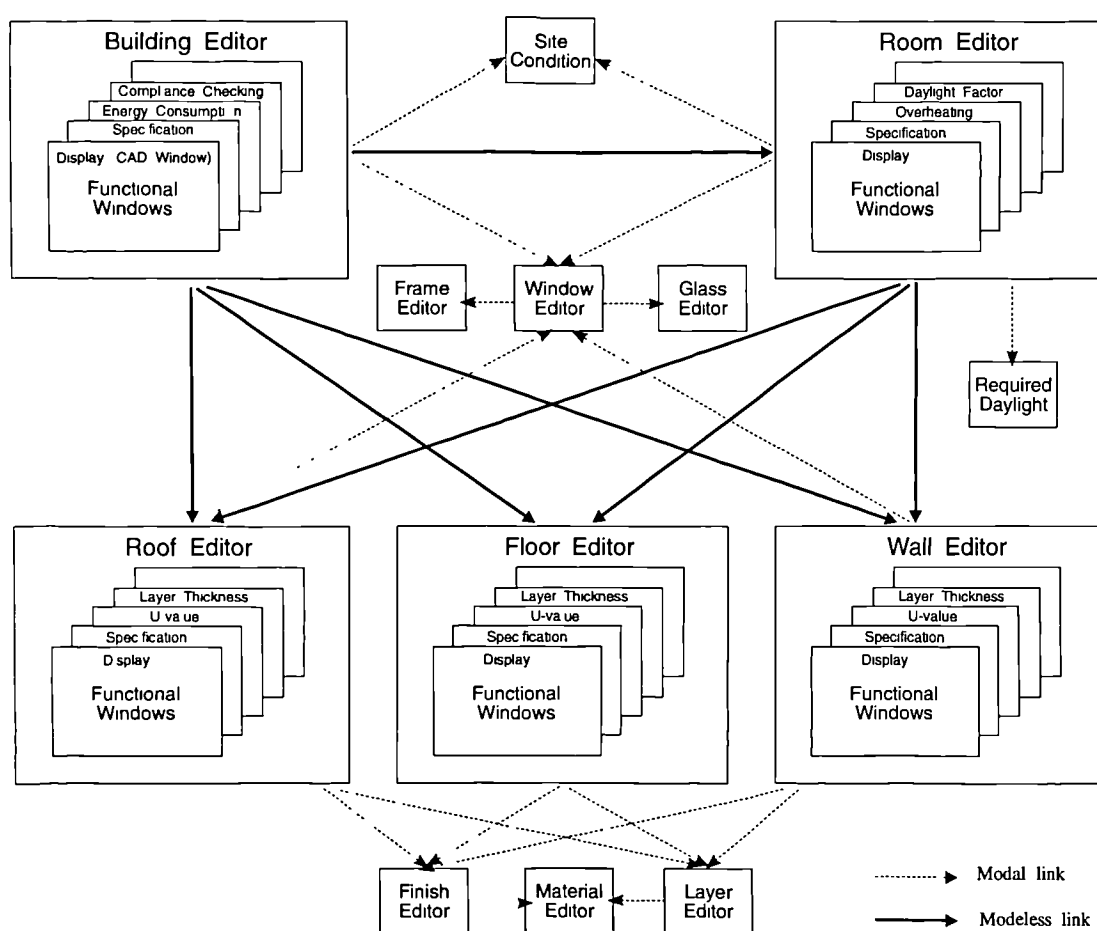


Figure 7.8 Interface structure of MultiCAD

In the figure rectangles represent user interface windows. There are two types of interface windows. One type is an application editor window including Building Editor, Room Editor, Roof Editor, Floor Editor and Wall Editor. These Editors are for the purposes of editing and evaluating complex objects which have application functions, for instance, energy consumption for buildings, daylight factor for rooms and U value for walls. Several application windows can run concurrently.

The other type of interface is dialog, such as Window Editor, Layer Editor and so on. The dialog interface is used to edit simple objects. These dialogs run exclusively. In other word, when one dialog runs, it disables its parent window.

Similarly, there are two types of links among the interface windows, namely, modeless link and modal link represented by solid and dotted lines respectively. The arrow on the line represents the sequence of the interface. A modeless link means two editor windows can run concurrently, a modal link means the dialog disables the editor window until the dialog closes. For example, the solid line between Room Editor and Wall Editor with an arrow pointing to the latter means that a modeless application window of Wall Editor can be opened from the Room Editor. In this case, the Wall Editor and Room Editor can remain operational at the same time. On the other hand, the dotted line between Room Editor and Window Editor with an arrow pointing to the Window Editor means that a modal dialog Window Editor can be opened from the Room Editor. The user, in this case, has to do the editing in the Window Editor then close it before go back to the Room Window. The objective of this arrangement is to build into the system a logic control to avoid possible confusion generated by too many interface windows being opened on the computer screen at one time.

The application editor window, such as building editor, room editor and wall editor, is designed as a Multiple Document Interface (MDI) window which allows multiple functions to be organised effectively. This guarantees that all evaluation applications for an object are always located within that particular editor. This is a further step to ensure the clarity of the system interface design. For example, if a U value calculation is conducted for two walls simultaneously, two U value windows appear on the screen. Without this kind of arrangement it is hard to tell which U value is for which wall. Figure 7.9 shows a standard interface application window.

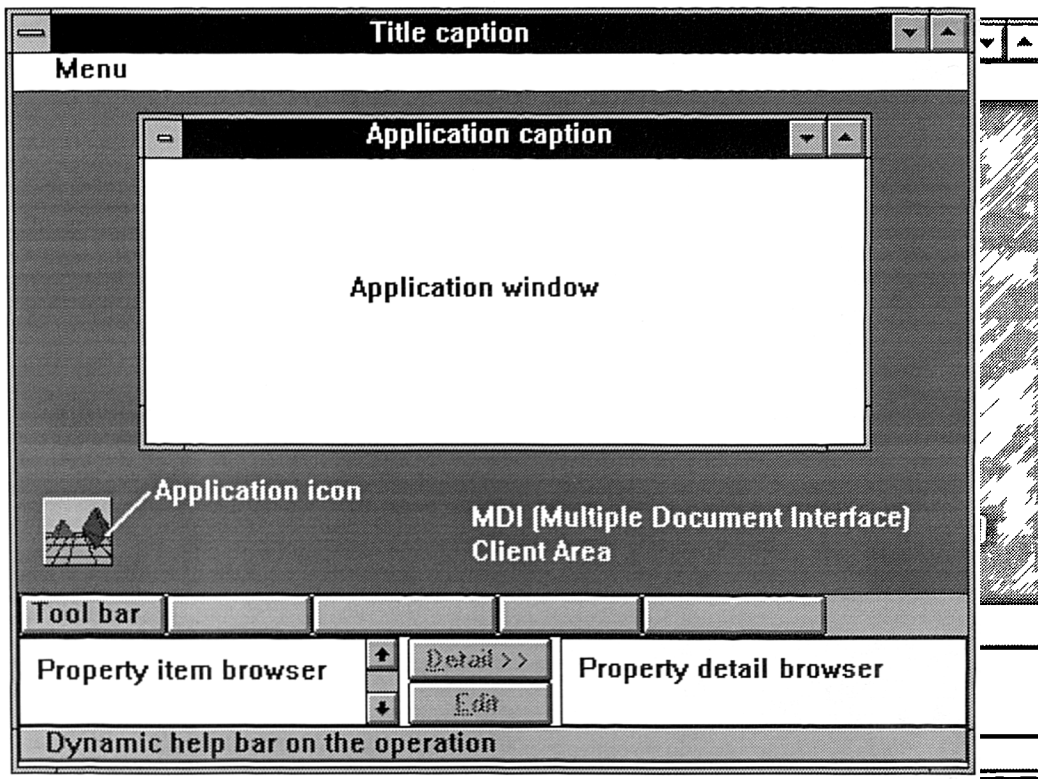


Figure 7.9 Standard application window interface

There are the following elements in this interface window. *Title caption* displays the editor name and the object it is working on for instance "Room Editor - bedroom 1". The *menu* groups the editing and evaluation functions. The *MDI client area* contains various application tools which can run concurrently. These *application tools* are fully integrated. The operation is managed by the Microsoft Window MDI mechanism. The *tool bar* consists of several buttons which are shorthand for the menu item for invoking application tools. There are two *browser* boxes which are used to display the properties of the design object. In addition, there is a dynamic *help bar* which provides brief help on the operation of the prototype.

The dialog editor, i.e., window editor, frame editor, material editor, etc., is implemented in the fashion shown by figure 7.10. The common components include *list boxes or combo boxes* which present a list of options, *edit boxes* which are used to obtain input. *OK* and *Cancel* buttons are used to accept or abandon the editing operation. Some addition buttons are used if necessary, for example, there is a button for editing material in the LayerEditor.

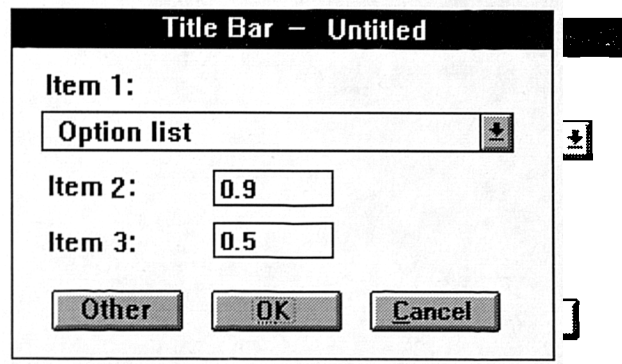


Figure 7.10 Dialog interface style

In the following the interface design is discussed for each individual editor. For each editor, the requirement is outlined first; then the design solution is presented.

Name: Building Editor

Requirement: The building editor should provide the user interface to instantiate and display the building object. It should be able to browse the properties of the building object and invoke evaluations such as building energy consumption, thermal compliance with the Building Regulation, performance analysis for rooms, elements and so on.

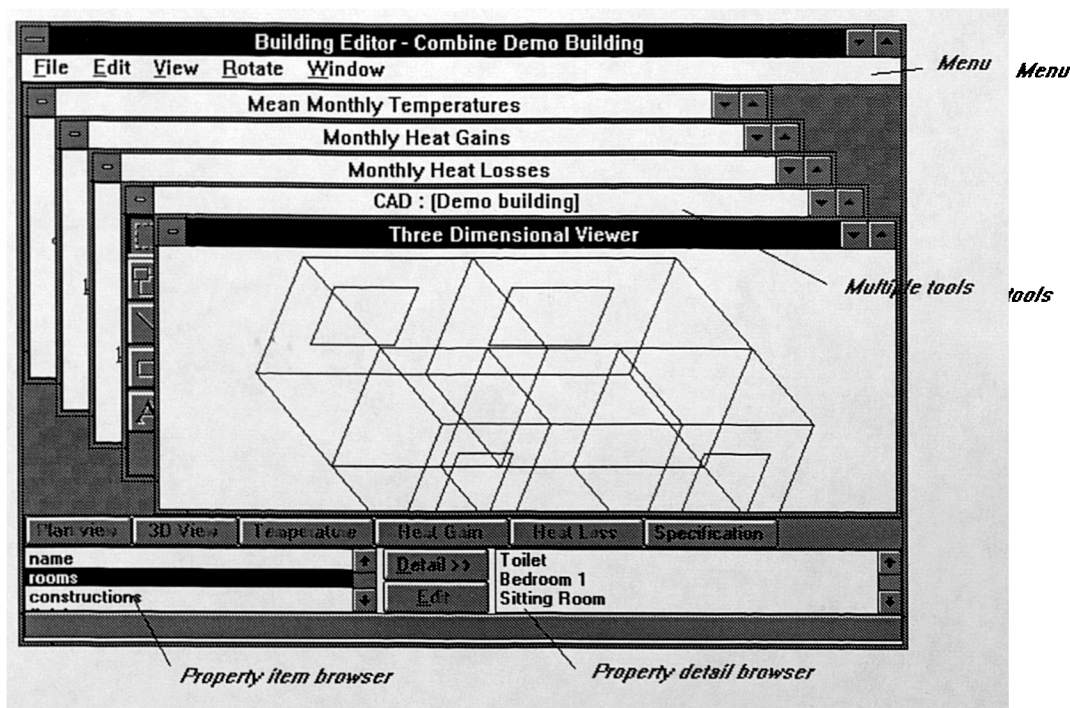


Figure 7.11 Building Editor

Design: The building editor is implemented as a MDI main window which contains several functional child windows, such as CAD window, 3 dimensional viewer, building

heat loss, temperature, etc. (figure 7.11). The CAD window provides an essential tool to draw the building plan in the drawing area. A 3D viewer provides a 3 dimensional view of the building. The data of the building can be browsed using the property browser provided. The evaluations on the building can be invoked using the tool bar or the window menu. The evaluation results are present graphically in separate windows. Any room or elements can be selected for detailed investigation from the building editor.

Name: Room Editor

Requirement: The room editor should provide a frame for the editing of the room parameters and the evaluation of room performances, i.e., daylight, overheating and so on.

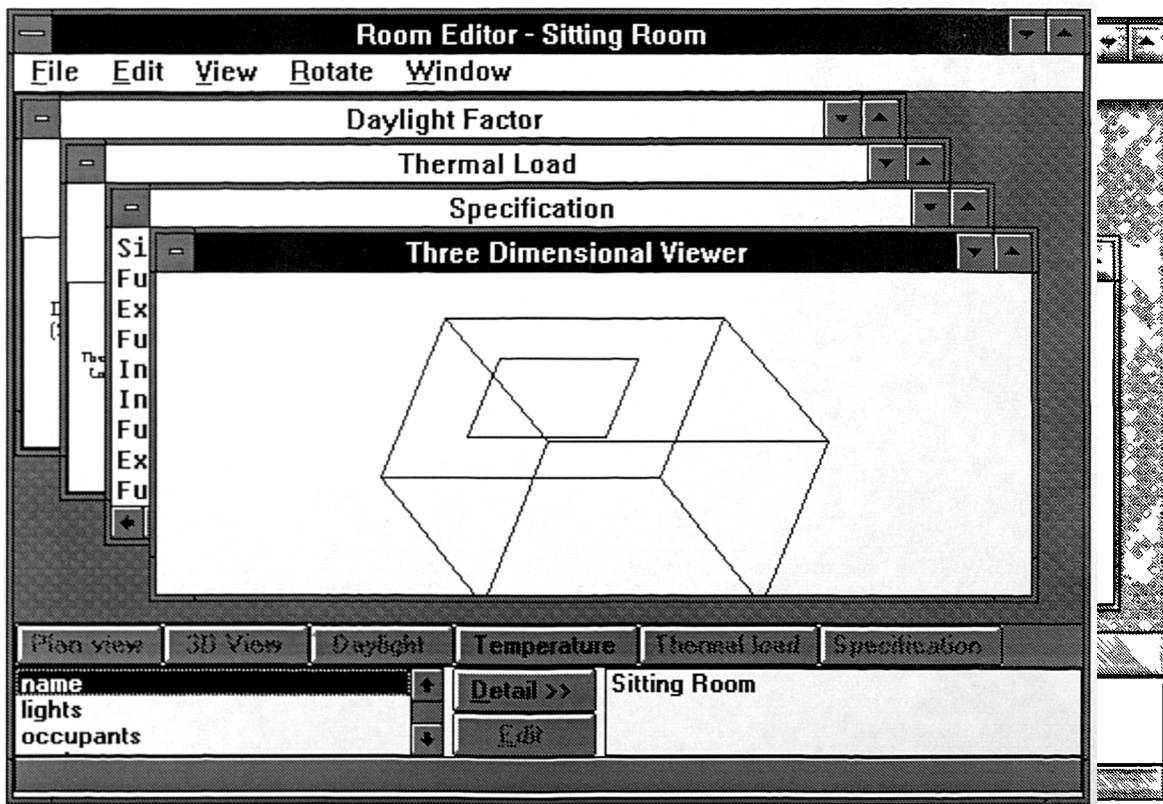


Figure 7.12 Room editor

Design: The room editor is designed as a MDI frame window (figure 7.12). It has a menu to manage the operation of the editor and various functions. A tool bar acts as a shorthand of the menu. The property browser is used to show the properties of the room and to invoke other editor windows. Daylight, overheating, thermal load and specification are designed as child windows displaying in the MDI client area of the Room Editor.

Name: Wall Editor, Floor Editor, Roof Editor

Requirement: The requirement for wall editor, (floor editor, roof editor) is similar to that of the room editor. It should provide a frame for the editing of the wall parameters and the evaluation against various criteria, i.e., u-value, condensation and so on.

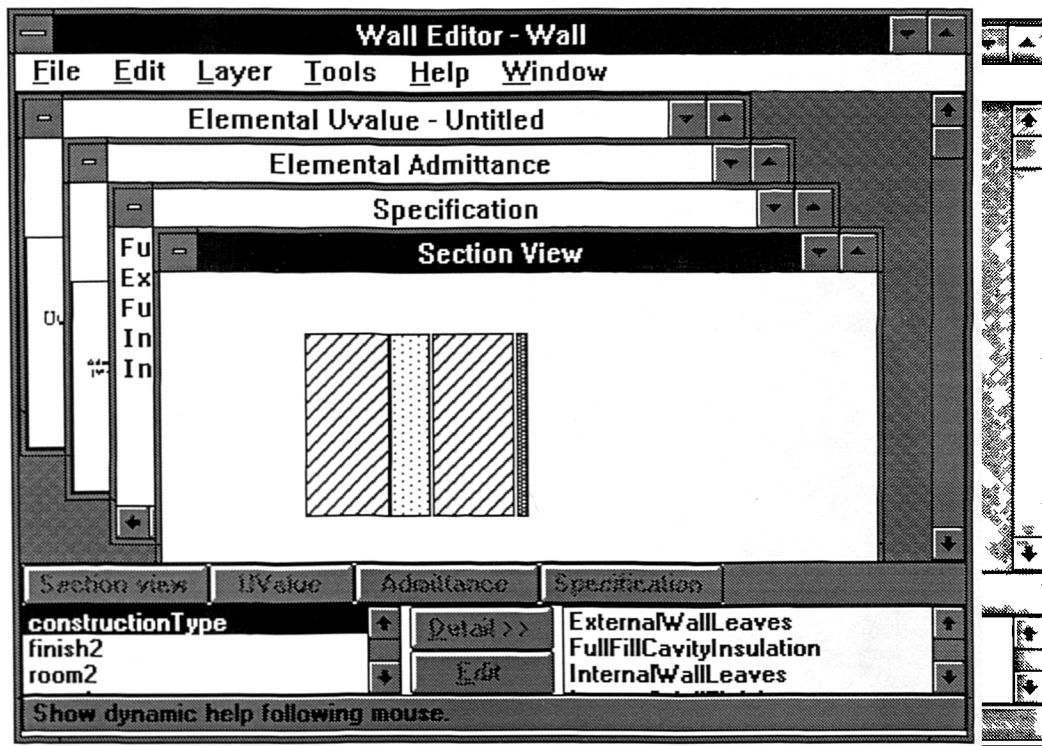


Figure 7.13 Wall editor

Design: The wall editor is also designed as a MDI frame window (figure 7.13). The operation principle of the wall editor is the same as that of the room editor.

Name: Site condition Editor

Requirement: This editor should allow the user to select one of the three options of the site condition in terms of maintenance factor for the window opening defined in the CIBSE Window Design Guide. In addition, the user should be able to decide the maintenance factor if he wants to.

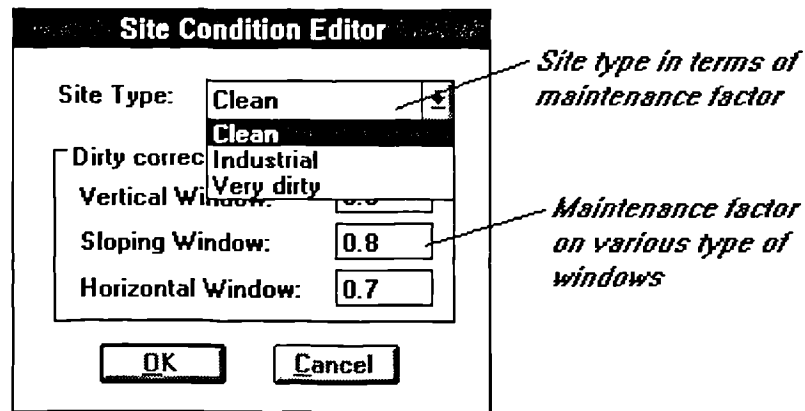


Figure 7.14 Site condition editor

Design: A combo list box presents the three options of the site condition for the purpose of daylight calculation (figure 7.14). The user selects one according to the condition of his design project. Then the default maintenance factors on various window openings are presented in the edit boxes. The user can make further changes to the value before confirmation.

Name: Window Editor

Requirement: The window editor should enable the user to edit the glazing type, glass material and frame type of the window. In addition, the visible sky angle needs to be inputted, which is used in the daylight calculation.

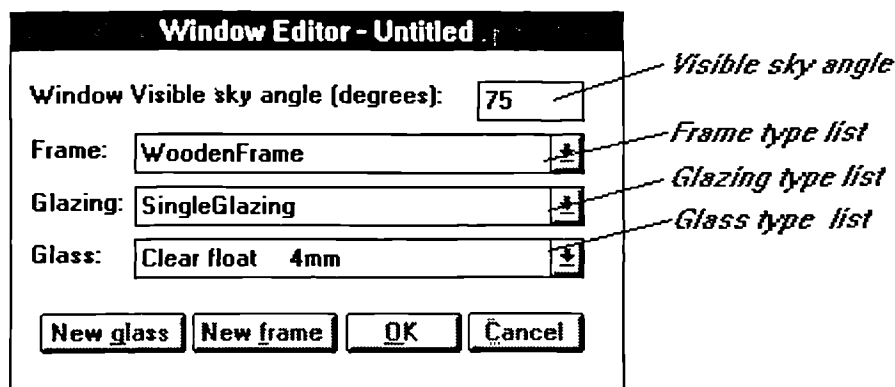


Figure 7.15 Window editor

Design: Three combo list boxes are designed for frame, glazing and glass material. The common used frame, glazing and glass types are listed respectively in these boxes. The

user can choose any combination. In addition, two buttons are used to input new window frame and glass material.

Name: **Frame Editor**

Requirement: The frame editor should be able to input a frame instance used in the data model, which has two instance variables, name and area percentage of the opening.

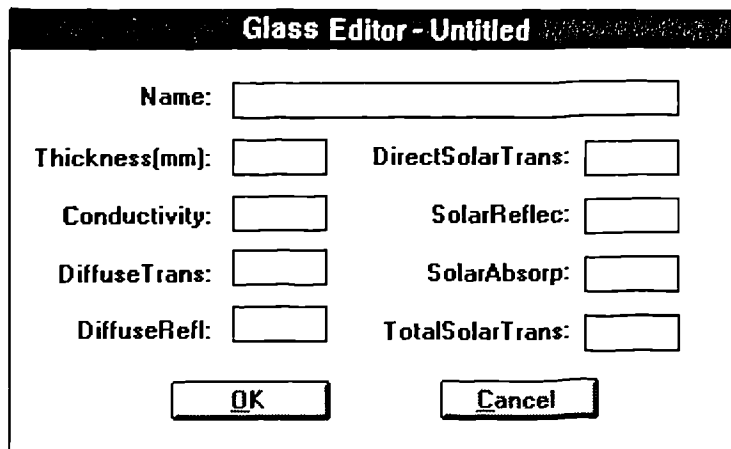
The image shows a screenshot of a software dialog box titled "Frame Editor - Untitled". Inside the dialog, there are two text input fields. The first field is labeled "Name:" and contains the text "Default frame name". The second field is labeled "Area percentage in the opening(%):" and contains the number "25". Below these fields are two buttons: "OK" and "Cancel". To the right of the dialog, there are two handwritten labels with arrows pointing to the input fields: "Frame name" points to the first field, and "Area percentage" points to the second field.

Figure 7.16 Frame editor

Design: Two edit boxes are used to prompt the user to put in the name of the frame and its area percentage in the opening.

Name: **Glass Editor**

Requirement: The glass editor is used to put in a new glass material or edit an existing one. The information needed for the glass material includes its name, thickness, thermal conductivity and so on.



Glass Editor - Untitled

Name:

Thickness(mm): DirectSolarTrans:

Conductivity: SolarReflec:

DiffuseTrans: SolarAbsorp:

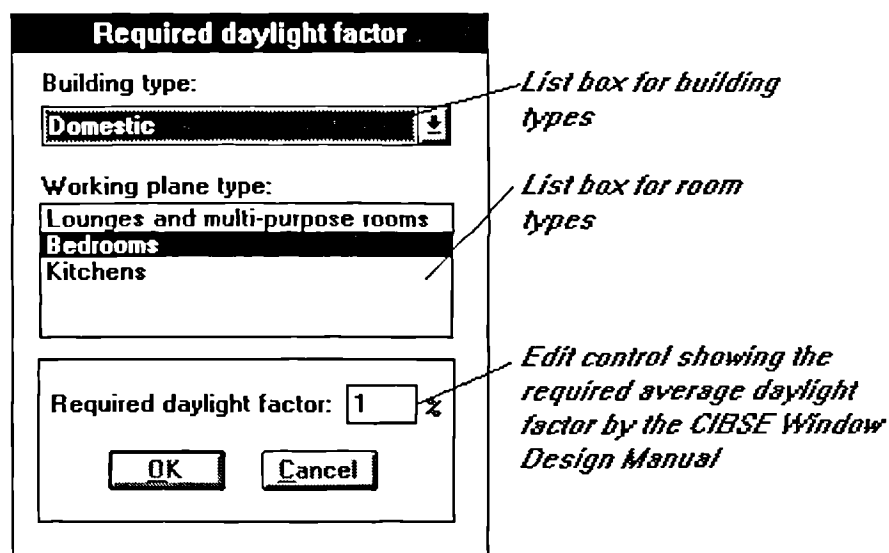
DiffuseRefl: TotalSolarTrans:

Figure 7.17 Glass editor

Design: Each item of the input information is handled by an edit box. Any illegal input will be warned and prompted for correction. This principle also applies to other dialog edit boxes.

Name: Daylight Requirement

Requirement: This dialog should be able to provide the information on the daylight requirement in the CIBSE Window Design Guide.



Required daylight factor

Building type: *List box for building types*

Working plane type:
 List box for room types

Required daylight factor: % *Edit control showing the required average daylight factor by the CIBSE Window Design Manual*

Figure 7.18 Daylight requirement dialog

Design: A combo list box is design to hold information of all the building types. The user chooses a building type from the list and the working plane options are presented in another list box. Once a working plane is chosen, the required average daylight factor is shown in an edit box.

Name: Layer Editor

Requirement: The layer editor should be able to edit the two elements of a construction layer, thickness and material.

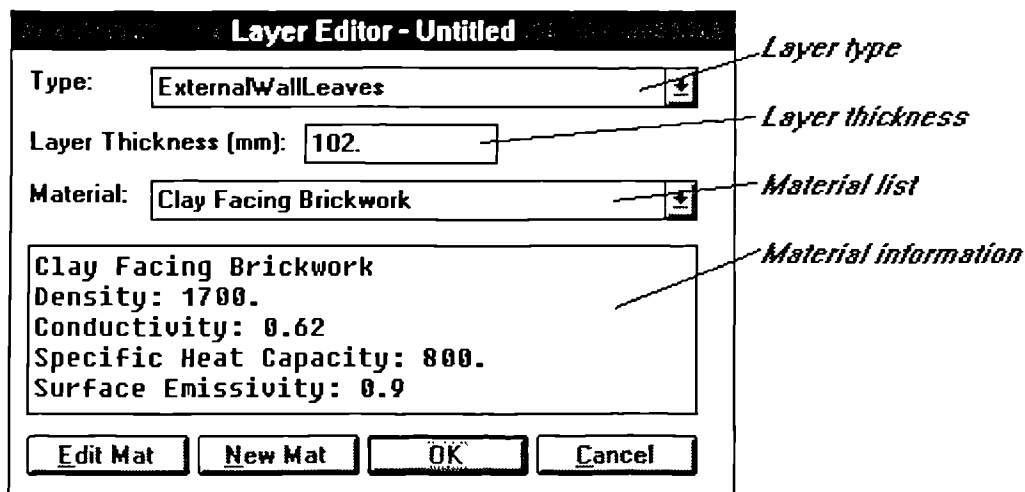
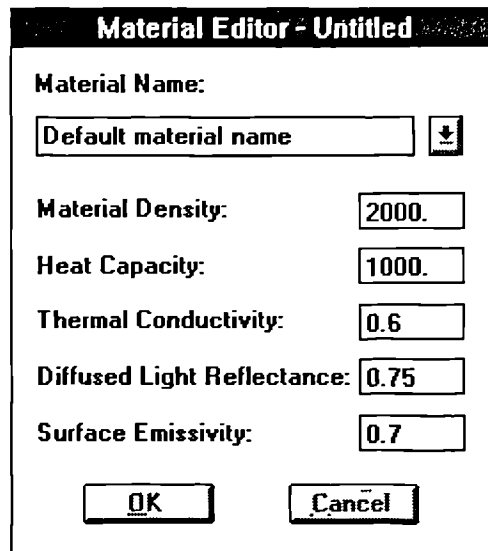


Figure 7.19 Layer editor


Design: An edit box is designed to input the thickness of the layer. A combo list box is used to present a list of options of the material. Essential information about the material is presented in a separate window. The user can change the specification of the material by clicking the "Edit Mat" button; or alternatively, input a new material by choosing "New Mat" button. In addition, this dialog provides a combo box for the common layer type. By specifying a layer type, appropriate material options will be suggested.

Name: Material Editor

Requirement: The material editor should be able to changing the specification of a material. Material attributes include name, material density, heat capacity, thermal conductivity, diffused light reflectance and surface emissivity.



Material Editor - Untitled

Material Name:
 

Material Density:

Heat Capacity:

Thermal Conductivity:

Diffused Light Reflectance:

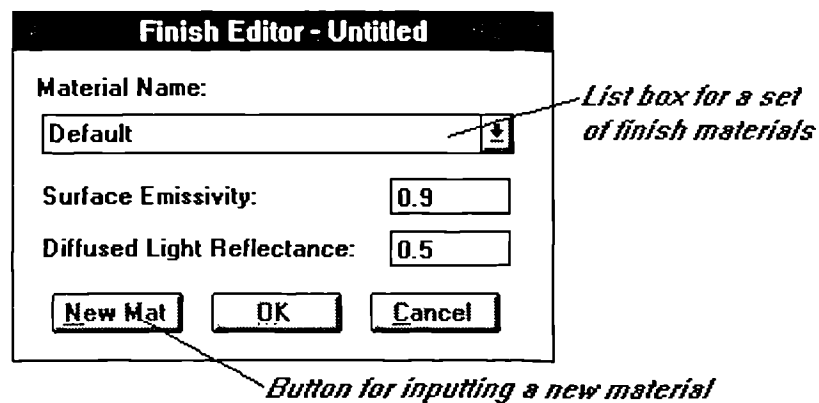
Surface Emissivity:

Figure 7.20 Material Editor


Design: The material name is handled by an editable combo box from which the user can choose an existing material name or type in a new name. Other properties are handled by edit boxes.

Name: Finish Editor

Requirement: The finish editor should be able to change the specification of a finish, including, its material, surface emissivity and diffused light reflectance.



Finish Editor - Untitled

Material Name:
  *List box for a set of finish materials*

Surface Emissivity:

Diffused Light Reflectance:

Button for inputting a new material

Figure 7.21 Finish editor

Design: A combo box is used to present a list of options of the finish material. Once a material is selected from the list, its default emissivity and light reflectance will be shown in two edit boxes. Further change is possible on these two properties. A "New Mat" button is provided for input a new material which is not in the material list.

7.8 Data Mapping with the IDM

Data exchange between the MultiCAD and the IDM is one of the major issues in the COMBINE project. The principle is illustrated in figure 6.4 which indicates that each DT maintains data exchange with the IDM. The figure also implies that there is no direct information exchange between different DTs. All the exchanges are realised off-line through the IDM.

The data exchange task requires the development of several separate programs using C++. The work is mainly done by Mr. Lockley, while the author cooperate from the part of MultiCAD.

The data model of MultiCAD is shown in figure 7.1. The IDM is a similar object oriented data model. Appendix C presents the EXPRESS schema of the IDM. The conceptual match between the MultiCAD data model and the IDM is indicated in table 7.1.

Table 7.1 Example of conceptual mapping between MultiCAD and IDM

MultiCAD		IDM
Building	↔	building_spatial_system
Room	↔	building_spatial_system.space[i]
Element	↔	space.enclosure_element[i]
ConstructionType	↔	enclosure_element.element_construction.construction_type
Layer	↔	construction_type.layer
Material	↔	layer.material

However, it has to be acknowledged that the conceptual mapping between these two models is not complete. The current IDM needs to be extended in the future in order to fully support the MultiCAD data model. Nevertheless, the data mapping exercise is realised to a very comprehensive extent.

For the purpose of demonstrating the data exchange principle, a school building is instantiated in the IDM. The IDM data is stored in ISO STEP file format, which is the starting point of the data mapping exercise between MultiCAD and the IDM. During the data mapping, MultiCAD extracts related data from the IDM, instantiates its OO building data model (Aspect Model), does some evaluations, makes changes to some data items and writes the changes back into the IDM. At present, MultiCAD does not read or writes STEP format data files seamlessly. Therefore, several translators are needed in this data mapping process. Figure 7.22 shows the configuration of this process.

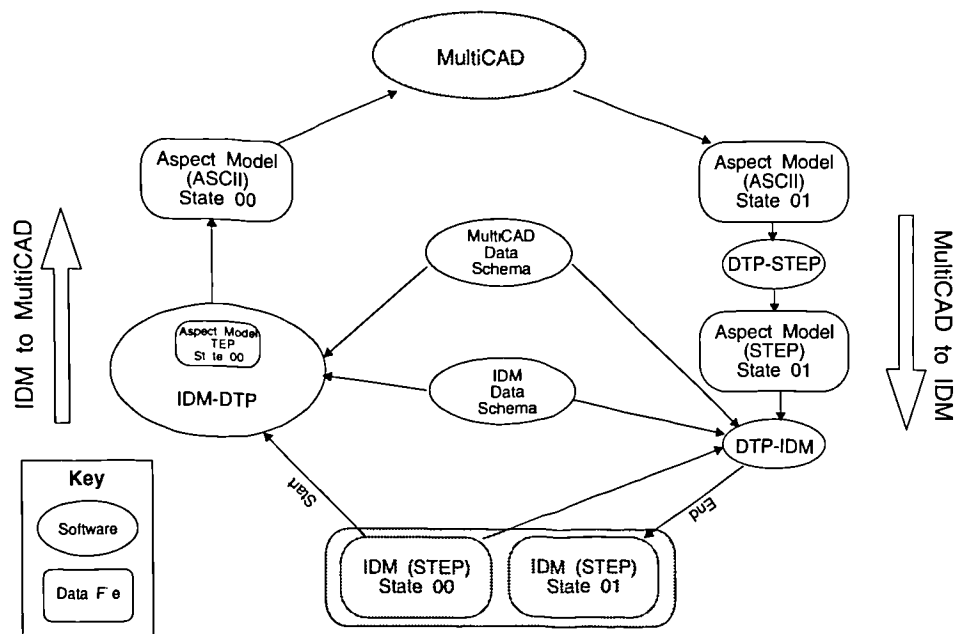


Figure 7.22 Data mapping between MultiCAD and IDM

One cycle of the data mapping consists of the following steps:

1. A translator (idmdtp.exe) reads in the original IDM STEP file (state 00) and produces the aspect model (state 00) of MultiCAD in ASCII format. It has to be pointed out that at this stage the MultiCAD aspect model is only partially instantiated since the IDM does not fully support the MultiCAD model.
2. MultiCAD reads in the state 00 aspect model and instantiates the full aspect model. Then, the MultiCAD editors can perform evaluations and data editing on the building. For the convention of the COMBINE demonstration, only construction data is changed by MultiCAD.
3. MultiCAD exports a new version of the aspect model (state 01) in ASCII format.
4. A translator (dtpstep.exe) converts the ASCII format aspect model into ISO STEP format.

5. Another translator reads in the new (state 01) aspect model STEP file and the old IDM (state 00) STEP file and produces an updated IDM STEP data file (state 01).

In general the integration of MultiCAD with the IDM using the STEP data exchange format has been successful. Figure 7.23 shows the visualisation of the IDM before the data mapping, interrogated using a tool supplied by the IDM team. Figure 7.24 shows the same building from the view point of MultiCAD after it is mapped to the MultiCAD data model. The geometrical and topological data as well as construction data of the example school building are successfully mapped from the IDM to the MultiCAD data model. Similarly, the data changes made during the MultiCAD operation are written back into the IDM without any problem.

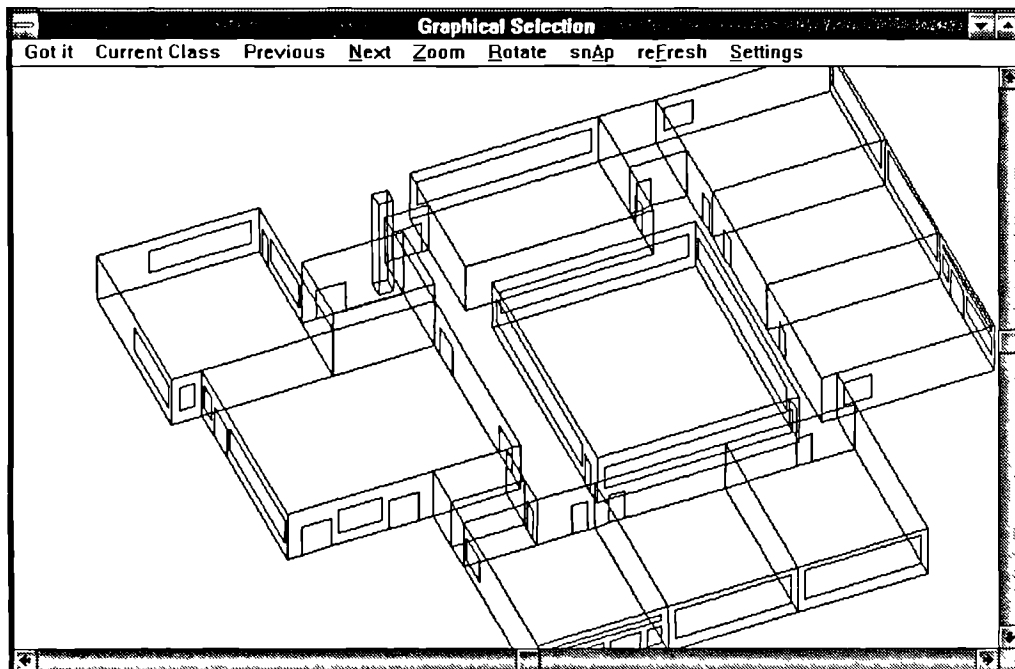


Figure 7.23 Visualisation of the IDM before the data mapping

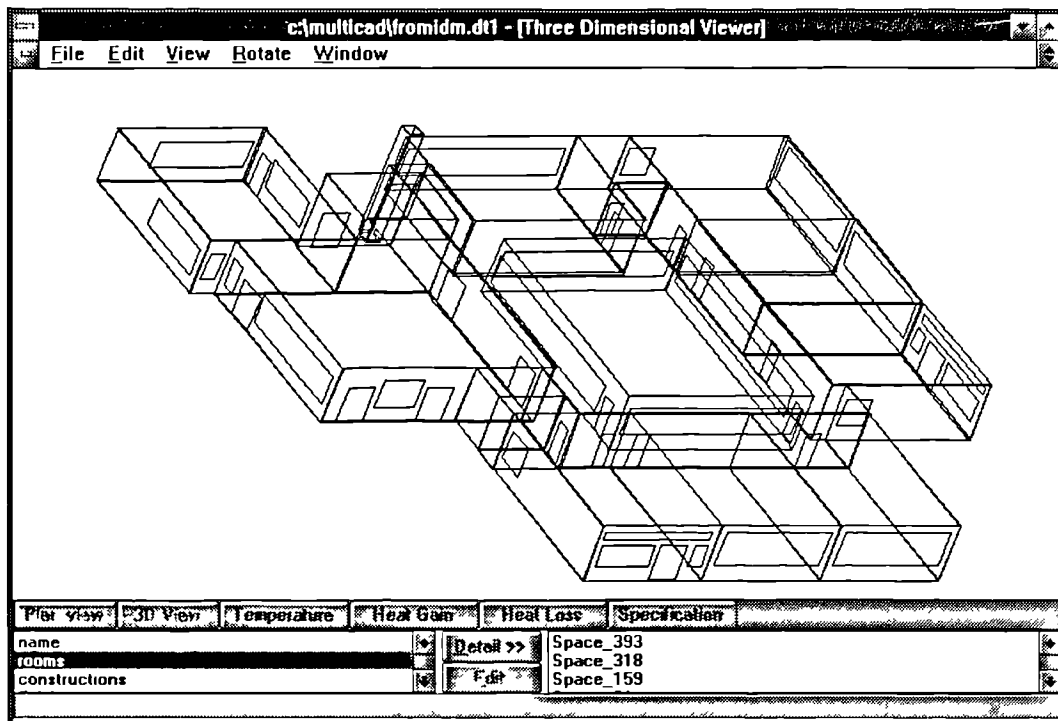


Figure 7.24 Visualisation of the MultiCAD data model after the data mapping

7.9 Coding Process

Coding is the process of transferring the system design concepts into programming code and producing executable programs. The chosen programming language has a major impact on the coding process and the software quality. Object oriented programming languages are believed to have major advantages over the traditional procedure language (Bell, 1992). ACTOR, the language chosen for the implementation of MultiCAD, is a fully object oriented language. It includes more than 120 classes of various types, i.e., numeric, collection, graphic, user interface windows, etc. These classes provide a rich functionality base for other application programs to be built upon.

The coding process in ACTOR includes the following steps:

1. Identify a class and create the class at a proper place in the ACTOR class hierarchy.
2. Specify the property or attribute of that class.
3. Implement the behaviour or methods of that class.

The implementation of MultiCAD covers two types of classes, (1) those of the static data model, such as Building, Room, Element; and (2) those for the purpose of user interface, i.e., BuildingWindow, RoomWindow, ElementWindow, etc. The first type defines the behaviour of the building and the latter type defines the interaction between the system and its user.

As pointed out above, the Actor programming language provides a rich class library which is ready to be used for the implementation of MultiCAD classes. Figure 7.25 shows the class hierarchy of the static data model of MultiCAD and its relation in the Actor class hierarchy.

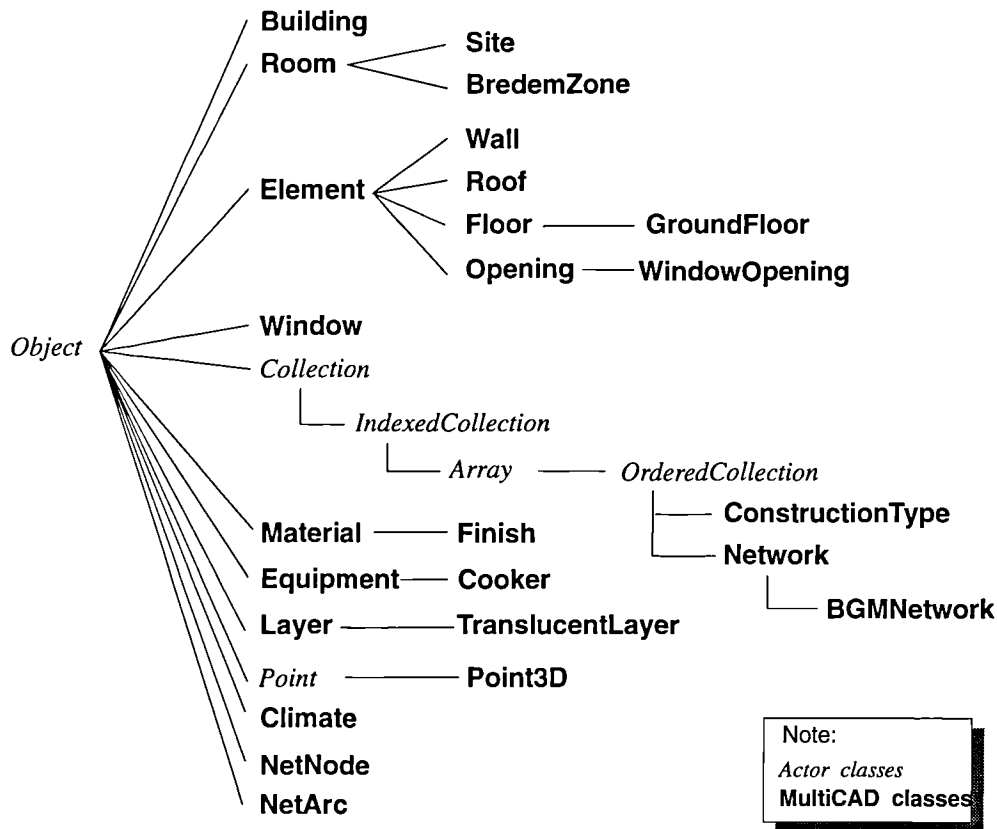


Figure 7.25 Class hierarchy of the data model classes

In figure 7.25, italic text represents classes provided by Actor and bold text represents classes defined in MultiCAD. It can be seen that all the MultiCAD classes are direct or indirect descendants of the top Actor class, *Object*, which provides functions general to all the classes, for example, creating new instances, copy an instance, memory management, etc. Descending from *Object*, all the MultiCAD classes can simply use these functions without refining them individually. The obvious benefit is the saving of development effort. The MultiCAD *Point3D* class is defined as a descendant of the Actor *Point* class which has already defined common attributes and behaviours of two dimensional points, such as x and y property, drawing self, drawing line to another point, moving and so on. The *Point3D* class only needs to define attributes and behaviours specific to 3 dimensional point, i.e., z dimension property, distance from another 3D point, etc.

Using the same principle, *ConstructionType* and *Network* are made descendants of the Actor *OrderedCollection* class. This type of arrangement demonstrates the inheritance principle of the Object Oriented programming.

Actor also includes many general classes for the purpose of generating an MS-Window style user interface, which has been fully exploited in the implementation of the MultiCAD user interface classes. Figure 7.26 shows the arrangement of the interface class hierarchy for the MultiCAD prototype.

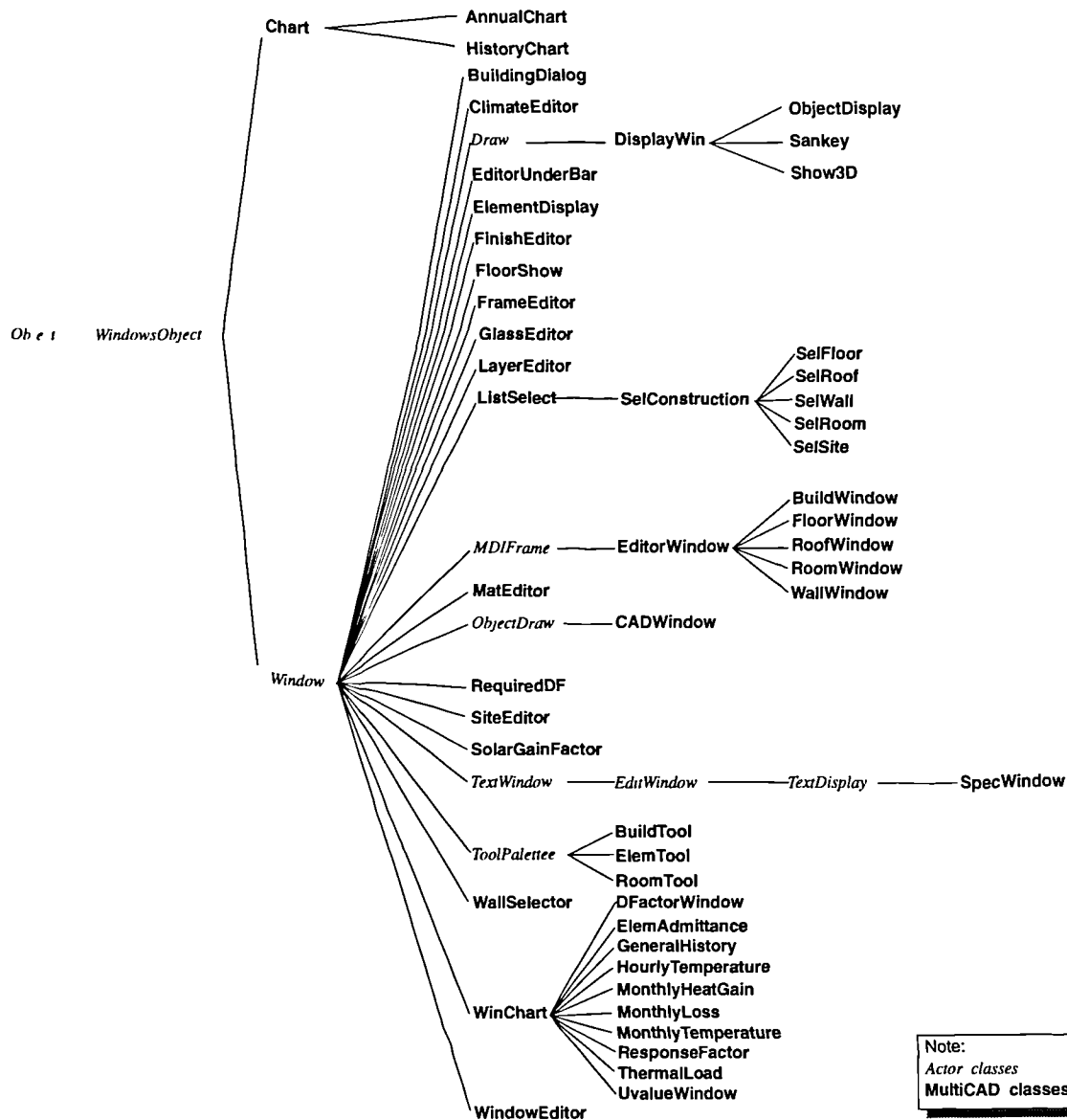


Figure 7.26 Class hierarchy for the user interface classes

MultiCAD is intended as an MS-Windows application which is an event-driven application. In other words, the system operates in response to events such as mouse moving, mouse clicking, keyboard pressing and so on. The interface of a typical Windows application consists of windows, dialogs, buttons, menus, etc. Actor, as part of its language, provides classes which can communicate with the MS-Window through a standard Application Programming Interface (API). These classes include *WindowsObject*,

Window, *MDIFrame*, *TextWindow* and so on. All the MultiCAD interface classes descend from these classes (figure 7.26).

In the following, an example is given to illustrate the purpose of the two types of classes, the data model class and interface class, and their interaction. The example is a daylight calculation for a room. It will show that the calculation method of daylight factor is implemented by the data model class, *Room*, but all the user interface is implemented in interface classes, *RoomWindow* and *DfactorWindow*.

Figure 7.27 is the MultiCAD user interface for Room Editor (*RoomWindow* class). It shows a three dimensional view of the room. The user starts a daylight calculation by clicking the *Daylight* button.

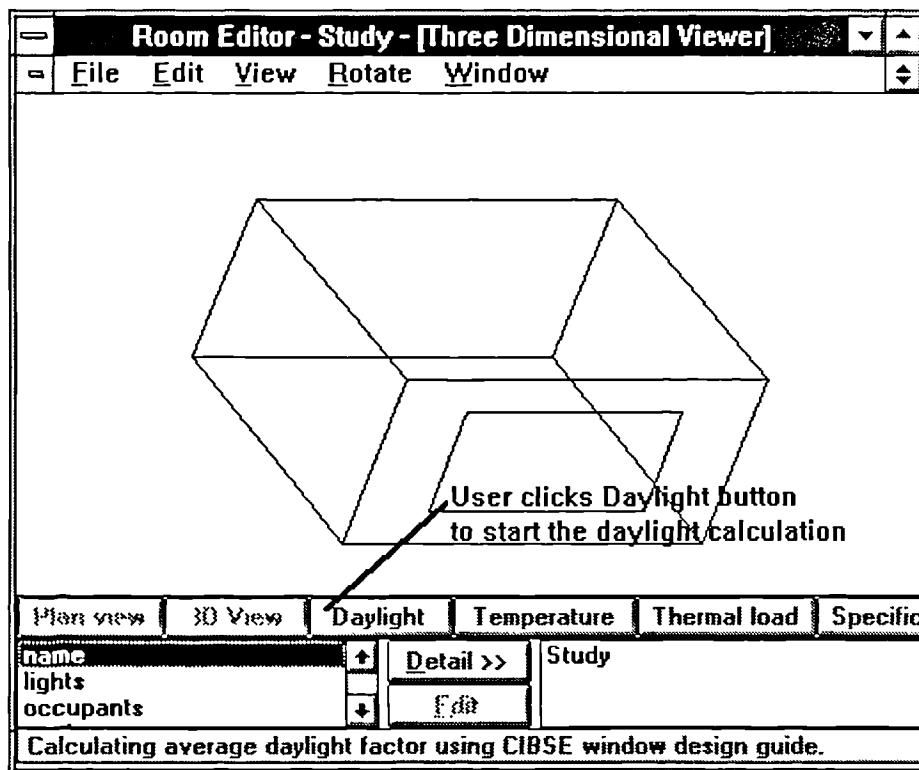


Figure 7.27 Room Editor user interface

The event of the user clicking the *Daylight* button triggers a sequence of message sending and method executing behind the user interface (figure 28).

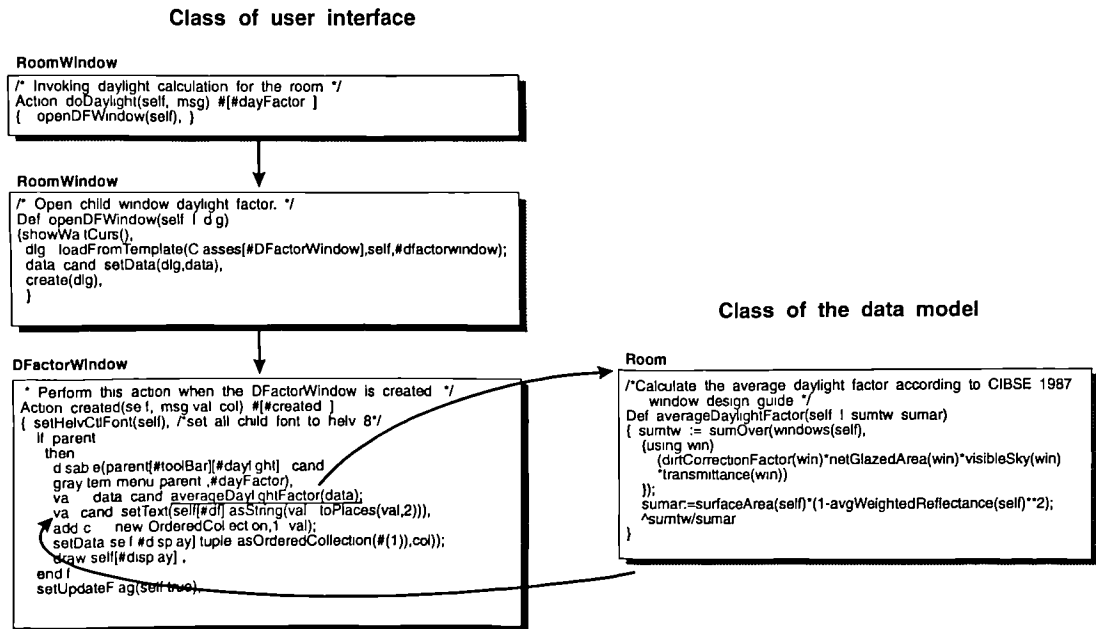


Figure 7.28 The sequence of the daylight calculation

The execution of the daylight calculation proceeds in the following steps:

1. When the user clicks the Daylight button (figure 7.27), the **doDaylight(self,msg)** action is invoked for the RoomWindow.
2. During the execution of **doDaylight(self,msg)** the method **openDFWindow(self l dlg)** is sent to the RoomWindow. The purpose of this method is to open a child window DFactorWindow.
3. When a DFactorWindow is called to create itself, the Action **created(self,msg)** is executed automatically. In the implementation of this method, a **averageDaylightFactor()** is sent to *data* which is the room under investigation.
4. When the room received the **averageDaylightFactor()** message, it knows its implementation. It executes the method and returns the result.
5. After getting the value of the daylight factor calculation, the DFactorWindow presents it properly to the user.

The above procedure is hidden from the user. Following clicking the *Daylight* button, the user will see figure 7.29 immediately, in which the DFactorWindow presents the average daylight factor for the room 3.57% in an edit box, at the same time a graphical comparison with a standard requirement 1.0% is given.

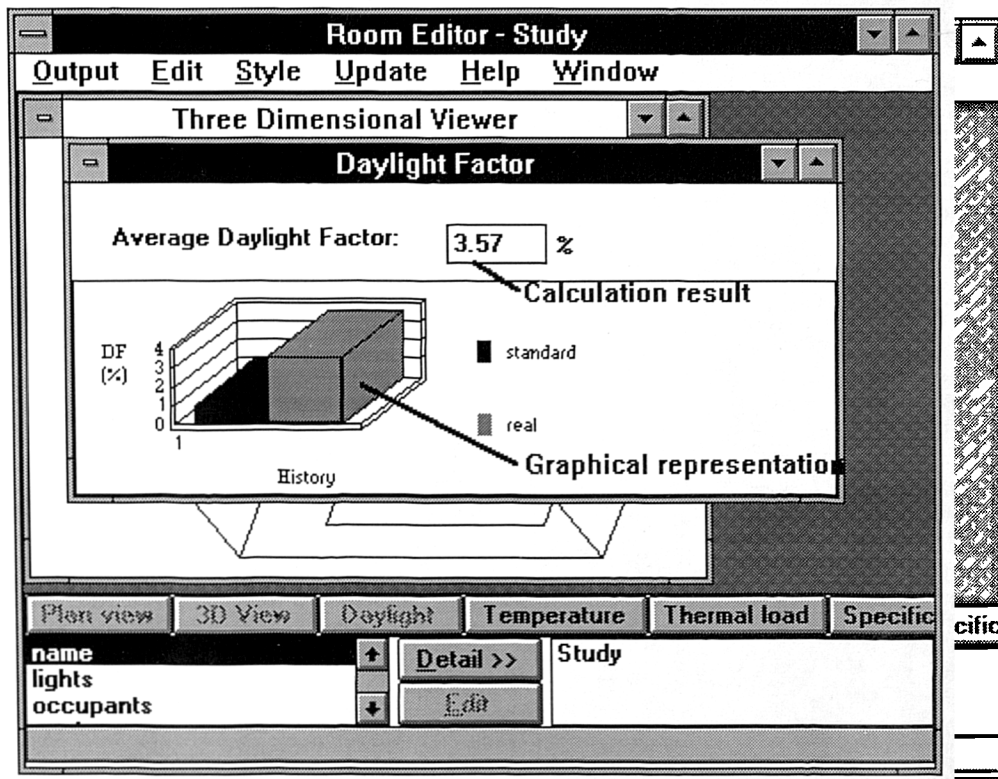


Figure 7.29 The interface for presenting the daylight calculation result

The Syntax of the coding in Actor programming language is shown in figure 7.28. The source code of a typical method includes several components, *comment*, *name* and *argument(s)*, *body*. The layout is as follows.

```
/* comment */
Def (or Action ) name (argument (s))
{ body }
```

The *comment* is to help the programmer and others to understand the purpose of the method. It is optional. The *name* is used when the method is called from other object or the object itself. The *argument(s)* field specifies the format in which the method is called. The *body* defines the content of the method.

There are two types of method in the Actor language, defined by **Def** and **Action** respectively. The difference between them is that the first type has to be call by name from other methods, while the latter type can be called by other methods or can be invoked by mouse and keyboard events. For example, the Action method **doDaylight(self,msg)** in figure 7.28 can be triggered by user choosing the menu item *dayFactor* or clicking *Daylight* button; the method **openDFWindow(self | dlg)** has to be called from **doDaylight** or other methods. Detailed documentation for Actor programming language is available in the Actor programming manual (Whitewater Group, 1991).

In the above example, many other methods are used, for example **surfaceArea(self)** and **avgWeightedReflectance(self)** for the room. All these methods are implemented separately, however the discussion here will go no further.

The source code for each class is stored in a file in text format. A class reference can be generated from the source code in rich text format. The information includes the ancestors and descendants of the class, its instance variables, class variables, class methods and object methods. For the methods, only name and argument format are given. The content of the code body is stored in a source code file. Nevertheless, such a class reference document can be useful for a software project. The following is a specimen of the class documentation for Room class. The documentation for other classes in the data model (but not the interface classes) is included in Appendix D. It shows the behaviours or methods of the objects in the data model, and it is also helpful in understanding the system when used in conjunction with the static data model information presented in section 7.5. Class reference for the interface classes, such as BuildWindow, FinishEditor, WallSelector, etc., is not included to avoid further lengthening of the thesis.

Room

Source file:	ROOM.CLS
Inherits from:	Object
Inherited by:	BredemZone Site
General space class	
Class variables:	(none)
Instance variables:	
id	For data mapping purpose. SM
name	name of the room
lights	number
occupants	number
equipment	equipment in the room
roomLinkers	from the building
rooms	rooms in the space

Class methods:

functions(self)
return the name of all the functions

readInVariables(self,obj,reader);
for data mapping with the IDM

Object methods:

addEquipment(self,equip)
add equipment to the room

addRoom(self,rm)
add a room to the space.

areaAdmittance(self)
total area weighted admittance of a room

averageDaylightFactor(self)
return the average daylight factor according to CIBSE 1987 window design guide

avgWeightedReflectance(self)
calculate area weighted average light reflectance

bpeMethods(self)
Specifies useful BPE methods, with no parameters

bredemZone(self)
room is in Bredem zone 1 or 2. Bredem-8 A.4 p6

doors(self)
return door objects of the room

editPicture(self,boundsRect)
return a plan picture

elements(self)
return a collection of all elements forming a room including those of rooms within rooms therefore this can be used to get all elements in a zone

equipment(self)
equipment

fabricLoss(self)
get all the elements bounding this room including sub-rooms and calculates the total fabric loss

finishes(self)
return finish of all elements of the room

firstExternalElement(self)
find an external element of a room

firstExternalWall(self)
find an external wall of a room.

floor(self)
return floor object of the room

floorArea(self)
temporary floorArea method. Checks for an element with z values all 0

getInfo(self)
getInfo

getPlanView(self)
return an ObjectGraphics Plan view

getSite(self)
get access to the site.

glazingLoss(self)
return heat loss from glazing in a room

hourlyAirToAirHeatGainSwing(self,hour)
modified for inf and multiplying by temperature

hourlyCasualGainSwing(self,hour)
hourly casual gain swing

hourlyEnvironmentalTemperature(self,hour)
hourly environmental temperature

hourlyInternalTemp(self)
calculate the hourly internal temperature for a room fro 24 hours in a day.

hourlySolAirTempSwing(self,hour)
hourly solair temperature swing

hourlySolarGainSwing(self,hour)
hourly solar gain swing

hourlyStructuralGainSwing(self, hour)
hourly structural gain swing

hourlySwingInEnvironmentalTemperature(self, hour)
hourly swing in environmental temperature

hourlyTotalSwingInHeatGain(self, hour)
hourly total swing in heat gain

id(self)
return identifier

information(self)
return brief information about the room

init(self)
initialise room object

initCasualSchedule(self)
for BREADMIT

isExternal(self)
return self if room is descendent of or is a site

isInternal(self)
return true if room is not external

isZone(self)
return whether a room is a single room or a zone

lights(self)
return a number

meanCasualGain(self)
mean casual gain

meanEnvironmentalTemperature(self)
mean Environmental Temperature

meanGain(self)
mean heat gain

meanInternalTemperature(self)
mean internal temperature

meanSolAirTemp(self)
mean solair temperature

meanSolarGain(self)
mean solar gain

name(self)
return name of the room

occupants(self)
return occupant number of the room

polyMesh(self)
return a polymesh that bounds the rooms volume

responseFactor(self)
response factor of the room

responseFactorAch(self, ach)
response factor of the room depending on air change rate

rooms(self)
return all the rooms if any

setName(self,id)
give a room an identity

setupRoom(self,elems)
addsa new room to the building. The argument is a collection of elements

setViewPlan(self)
modified plan view which stores graphic links

solarGain(self,building,month)
solarGain Bredem-8 A.4.3 p9-10 in kWh/day

specificHeatLoss(self, ach)
specific heat loss

storeDefinitionOn(self, stream, table)
for data mapping

surfaceArea(self)
surfaceArea in square metres

thermalLoad(self,ach,to,ti)
thermal load of the room

totalConductance(self)
total conductance

totalInternalAdmittance(self)
total Internal Admittance

totalSwingInHeatGain(self,hour)
total Swing In Heat Gain

ventilationLoss(self,ach)
ventilationLoss Bredem-8 p7

volume(self)
calculate volume of the room

wallArea(self)
return the total wall area for STEP data mapping.

walls(self)
return a collection of all elements forming a room including those of rooms within rooms therefore this can be used to get all elements in a zone

windows(self)
return all the window openings of the room

Actor provides a very friendly software development environment. The coding, compiling and debugging can be carried out interactively. Object instances are easily created during the coding process for testing. The behaviour of each class can be implemented incrementally.

Another advantage of using object oriented language is that it enables the coding of a system to be done independently by different programmers. As long as some conventions are observed, the final integration of the system is not difficult. This has been demonstrated in the implementing process of MultiCAD, where five programmers have been involved in the coding process to various degrees.

7.10 Discussions on the Implementation

Efforts are made during the implementation to make the MultiCAD more friendly to architect users. The required input is expressed in architectural language. For example, typical input required from the user is construction material, glazing type, window frame type, etc. A small database is built in the system which enables the user to select, from lists, data such as, element construction type, layer type, material type, glazing type, window frame type and so on. Changing of the specification of the supplied data, i.e., conductivity of a material, is easily done by using the relevant editor, material editor in the case of material (section 7.7).

Another strategy to improve the efficiency of the MultiCAD operation is by providing default data. In other word, when a building or any other object is instantiated, it is created using default data. It is believed to be better to provide default data and allowing users to overwrite than to ask the user input every data from scratch. At the early design stage, some of the data items concerning the building may not have been decided. By accepting the default, the user can proceed with the performance evaluations. The user can change the default data later on and the evaluation results will be updated accordingly.

The current implementation of MultiCAD largely satisfies the requirements set out at analysis and design stages, this will be demonstrated in next chapter's description on testing and evaluation. However, since it is a prototype process, there are areas need further work.

The design request analysis of section 6.5 has identified four generic types of request, (1) general guidance on a design aspect, (2) evaluation of a design solution against specific criteria, (3) sensitivity test, examine the impact of a parameter(s) on the result, (4) value of a design parameter to meet a design target. These requests are written into the functional requirements of MultiCAD (section 6.6). *The first three* types of request have been fully implemented in the current MultiCAD prototype.

However, the implementation of the fourth type has not been straightforward. In some cases, it can be done, for example to calculate window opening area for a required daylight level or the insulation layer thickness for a required U value for a wall. Both of these two cases have been implemented in a *daylight calculator* and a *uvalue calculator*. In other cases, the algorithm is too complex to perform in two directions. For example, it is almost impossible to calculate the window opening area required to achieve a building energy consumption target using BREDEM-8. In these cases, the user relies on quick trials to get the required value for a design parameter, to aid this process MultiCAD provides a graphical comparison between the calculated value and a standard value for most evaluations.

In figure 7.5, a **blackboard control** is shown in the system structure, which acts as a check control between an evaluation process and the data model. This requires taking a temporary copy of the object under investigation. In the *Actor* programming environment, this is done by a deepcopy function. If the object is a complex one, i.e., a building which contains rooms, elements, construction types, layers, ..., the deepcopying is done recursively to all the levels. As a result, a large amount of objects are created in memory which cause problems to the system. This issue can be addressed by the object versioning mechanism in an object oriented database system (Kim, 1990a). Integration with an object oriented database is the objective for the next phase development of MultiCAD. Therefore, **blackboard control** is not implemented at the current version of MultiCAD.

Another area left to be solved by the object oriented database is the object persistent storage. At the moment, all objects created during the system operation exist in computer memory, they are destroyed when the program terminates. MultiCAD only provides a limited saving facility to persistent memory. The future integration with the database will solve this problem. No further effort is thought to be necessary in this respect at this stage.

The CAD facility of MultiCAD is purely for the purpose of prototyping. It is sufficient to interrogate the geometry of a simple building. However, it has limitations in handling complex buildings, i.e., buildings with irregular shaped rooms, or when a room position needs to be changed. The integration with a commercial CAD system is believed to be essential before MultiCAD can be put into practical use in the architectural design offices.

7.11 Summary

This chapter explains the design and implementation process of MultiCAD prototype. An Object Oriented (OO) paradigm is adopted for this process. The data centred OO approach in software design and implementation has advantages in several important aspects, i.e., modularity, efficiency, extendibility, reusability, etc.

The building data model is considered to be the core of the prototype. The model consists of 25 object classes which correspond to real world entities, such as building, room, wall, and so on. Each of these classes specifies its data structure as well as the relationship with other object classes. In an OO paradigm, functions are dynamic behaviour which is defined by object class methods. In MultiCAD, energy consumption is a behaviour of a building objects; daylight is a behaviour of a room; u-value is a behaviour of an element. The methods for all the classes in the data model are documented in Appendix D.

Another important part of the system is its user interface. MultiCAD is designed as an MS-Windows application. Its interface design and implementation try to meet the requirements of an integrated, interactive and user friendly building design system.

CHAPTER 8

PROTOTYPE DEVELOPMENT, TESTING AND EVALUATION

8.1 Introduction

This chapter discusses issues concerning software testing and evaluation also known as software verification and validation (V&V will be used for short in the following text). They are quality assurance measures during the software development process. Boehm (1978) in a widely quoted statement defined verification and validation as "Verification: are we building the product right? Validation: are we building the right product?" In other words, verification is to ensure the software performing its functions correctly, and validation is to ensure the software functions meet the needs of its users.

The chapter addresses issues of two areas:

- an overview of the software V&V in general, and
- some V&V activities of the MultiCAD prototype in particular.

8.2 Overview of Software V&V

In the last half a century, computing technology has been entering human life at an astonishing pace. Today, in many important fields, such as aviation, military, telecommunication, banking, commerce, manufacturing, etc., we have to rely on the help of computing facilities. Therefore the reliability of computer software is crucial to the success of some missions with huge stakes of economic loss and even human life. On the other hand, software, as a creation of human individuals, inevitably includes defects which might cause failures under certain circumstances due to the existing technological limitations and due to human error. In order to prevent or minimise these possible defects, continuous testing and evaluation are needed during the software development process.

8.2.1 What is V&V?

V&V is one of the software engineering disciplines, which helps to build quality into computer software. It emerged in the late 1960s, initially for the development of military and nuclear-power systems. The objective of software V&V can be broadly defined as the evaluation and testing of software in order to determine that it performs the intended functions correctly, to ensure that it performs no unintended functions, and to measure the software's quality and reliability (Wallance, 1989). Verification and validation are two different concepts.

- Verification involves evaluating software during each life-cycle phase to ensure that it meets the requirements set forth at the previous phase.

- Validation involves testing software and its specification at the end of the development to ensure that it meets users' requirements.

8.2.2 Why is V&V needed?

"If a system can go wrong, it will." This famous "Murphy's Law" also applies to computer software. The increasing significance of computer software in modern society underlines the importance of its reliability. Unfortunately, in reality no software is absolutely safe. Potential errors could be introduced into a software system at any stage of its life cycle. The benefit of constant V&V is to uncover the system defects at the earliest opportunity in order to fix them at a minimum cost.

Software development is a very complex engineering process. There are many influencing factors, such as user requirements, hardware, software constraints, the software developers themselves and so on. Software quality requirements are also complex and multiple facets. There could be a long list of requirements for the software development, i.e., correctness, robustness, extendibility, reusability, compatibility, usability, maintainability, etc. The comprehensive quality requirements make the regular conduct of V&V a necessity. Today, no serious software products could be released without appropriate V&V activities. Pressman (1987) suggested as much as 40% of the efforts could be spent on the system V&V during a typical software development process.

Wallace (1989) outlined several benefits for the regular V&V activity during the software development:

- It helps to identify the right system requirements by task analysis at the beginning of the development.
- It evaluates the software against the system requirements to guarantee the fulfilment of the specification set forward.
- It uncovers high-risk errors early, and allow the development team to fix them promptly.
- It improves the productivity of the software development.
- With the users' involvement in the system development process, it gives the users an incremental preview of the system performance, with the chance to make early adjustments.

8.2.3 When should V&V be applied?

Figure 6.1 in Chapter 6 showed the classical life cycle of a software product. It illustrates the phases of analysis, design, implementation, system test and maintenance. At each

milestone of this cycle, system V&V can be conducted to eliminate possible errors which might occur during the previous phase.

Fairley (1985) pointed out three major categories of software error, requirement errors, design errors and implementation errors.

- Requirement errors are caused by incorrect interpretation of the user needs, by failure to correctly specify functional and performance requirements, by inconsistencies and contradictions in the system requirement specification, and by unfeasible requirements.
- Design errors are introduced by failure to translate the system requirements into correct and complete solution structures, by inconsistencies within the design specifications, and by inconsistencies between the design specifications and the requirements.
- Implementation errors are made in translating design specifications into source code.

Since errors can be introduced to the software at any stage, various forms of quality control measures are needed to minimise or remove them. Figure 8.1 shows the V&V activities in the software development process (Fairley, 1985; Smith, 1987; Deutsch, 1988).

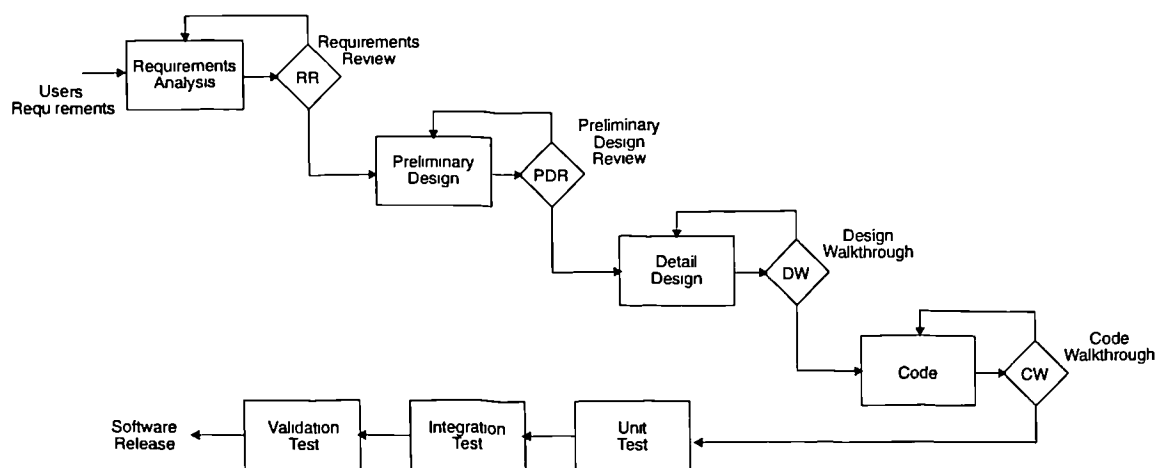


Figure 8.1 Software development and V&V activities

The tasks at each stage are as follows:

- Requirements review is conducted after the requirement analysis. It verifies the requirement specification against the criteria such as correctness, completeness, consistency, accuracy and so on.
- Preliminary design review assesses the translation of requirements into software design. The focus at this stage is on the software architecture. The concerning issues include: "Whether the software requirements are reflected in the software

architecture?" "Has the interface between the software and external element been defined?" "Has the interface between system modules been established?" "Is the data structure consistent with the information domain and software requirement?".

- After the detailed design being finalised, design walkthroughs are needed to ensure the validity of the software design. The evaluation criteria again include correctness, completeness, consistency, accuracy and so on.
- A code walkthrough is an effective means for uncovering errors made during the coding process. Typical errors include misspellings and typos, violations of the programming language conventions, incorrect data declaration and so on.
- During the unit test, small units of the software, i.e., a procedure or an object method, are tested separately in great detail. The objective is to ensure the correctness of all part of the software.
- After the unit test, the development process enters a new phase, system testing. System test covers two broad tasks, integration test and validation test. Integration test assesses the software working as a whole against its specification. Validation test measures the software against the users' requirements. Therefore, it tests the software specification as well as the quality of the construction of the software.

8.3 Techniques of Software V&V

There are a large number of V&V techniques and tools and Wallance (1989) listed more than forty of them. She argued that the choice of a particular V&V technique depends on the characteristics and V&V objectives for each project under consideration. These V&V techniques can be broadly divided into two groups, review and testing. Some people single out a further type, audit (Poston, 1986a). As Poston (1986b) explained, the difference between review and audit is the nature of the participants not the techniques being used.

Software review usually involves a meeting of a group of people examining the progress at each stage of the development process. Walkthrough and Inspection are two commonly used software review techniques.

Testing is usually conducted after the implementation of the software but before its release. It is a series of quality assurance activities performed by executing the software against its requirements.

8.3.1 Walkthrough

Walkthrough is used to systematically examine the software under development throughout its life cycle. In a typical walkthrough session, there is one reviewee and

several reviewers. The reviewee is normally the person who is responsible for the work being examined, the reviewers are colleagues or experts in related areas. The reviewee presents or "walks through" his/her work products. Reviewers evaluate the work and raise questions on issues of concern.

The aim of walkthrough is to discover problem areas within the material to be examined. Although suggestions on solving the problems might be made during the walkthrough, solving the problems is the responsibilities of the reviewee after the session. Fairley (1985) suggested several guidelines for the conduct of walkthroughs in order to make them effective. These guidelines are summarised as follows:

- (1) Everyone's work should be reviewed on a scheduled basis. A one-off session will not possibly solve all the problems.
- (2) Emphasis should be placed on detecting errors during the review session. Participants should not spend too much time on problem solutions, because it is not an effective practice. All the issues raised during the session should be recorded in detail.
- (3) Major issues should be addressed as priority. Although it is difficult to distinguish major and minor issues, the guideline is that problems which can be detected and fixed easily by the reviewee alone should not attract too much attention.
- (4) There should be a time limit for each walkthrough session, say 2 hours. This helps to limit the scope of material to be examined and to concentrate the effort on major issues.
- (5) The walkthrough session should be conducted in a positive and non-threatening atmosphere. It should not be considered as employee evaluation.

Walkthrough is a fairly easy yet effective means to evaluate the software products. If it is conducted by colleagues working on the same project, it helps everybody to understand other's work more fully. Therefore, a regular walkthrough of each other's work can improve the effectiveness of the co-operation, productivity and quality of the development process.

8.3.2 Inspection

Inspection was first introduced by Fagan (1976). It is similar to walkthrough. However, the difference is that inspection is done by specially trained inspectors with specific roles to play during the inspection session. They inspect the work products against a pre-defined checklist. The inspection items vary depending on the work to be inspected and the stage of the software development. For example, at design stage the inspection item might be completeness of the design with respect to the user and functional requirements,

consistency in definition and the usage of terminology, and the correctness between modules. At the implementation stage, the inspection items would include subsystem interfaces, data referencing, data flow, computational expressions, I/O statements, memory usage and so on. Figure 8.2 is an example of inspection checklist at requirement analysis (Ackerman 1989).

<p>Completeness</p> <ol style="list-style-type: none">1. Are all sources of input identified?2. What is the total input space?3. Are there any 'don't care' sets in the input space?...17. Should reliability requirements be specified? Is an operational profile specified? <p>Ambiguity</p> <ol style="list-style-type: none">1. Are all special terms clearly defined?2. Does each sentence have a single interpretation in the problem domain?3. Is the input-to-output mapping clearly defined for each type of run? <p>Consistency</p> <ol style="list-style-type: none">1. Do any the designated requirements conflict with the descriptive material?2. Are there any input states that are mapped to more than one output states?3. Is a consistent set of quantitative units used? Are all numeric quantities consistent?

Figure 8.2 Sample checklist for requirement inspection

An inspection team ideally consists of four people, each of whom plays one of four well-defined roles, moderator, designer, implementor, and tester. According to Fagan (1976), the moderator is the key person in an inspection process, whose duties include scheduling the meeting, controlling the meeting, reporting the inspection results, and following up on rework. For best results, the moderator should be specially trained. Designer is the person who produces the design of the program. Implementor is the person responsible for physically implementing the design. Tester is responsible for the organising of the testing. These are the roles in the inspection session and the actual members may change depending on the product being examined. A complete inspection process will include the following elements:

- (1) Overview (whole team). This is a meeting where the designer of the product describes the details of the products, i.e., the architecture, processing logic, database, interface and so on. The purpose of this meeting is to get every member of the inspection team familiar with the product under examination as well as the areas of inspection. Documents are normally distributed at this meeting.

- (2) Preparation (individual). The participants individually study the documents concerning the product and go through the requirement checklist on which the inspection items are listed.
- (3) Inspection (whole team). During this inspection meeting, one of the participants, normally the implementor, is designated as the "reader" by the moderator. The "reader" paraphrases the design as expressed by the designer. The other participants follow the "reader" and raise questions when necessary and pursue these to a point where an error is found and recorded.
- (4) Rework. The designer and implementor rectify all the errors that were discovered during the inspection meeting.
- (5) Follow-up. The objective of this step is to verify the rework.

Fagan gave several examples to prove the effectiveness of inspection. In one experiment, 67% of the errors identified during the development were found before any unit testing was performed. Furthermore, during the first 7 months of operation, 38% fewer errors were found in the inspected program than in a similar program produced by control group using walkthroughs alone. The productivity of these two teams was about the same. There are other reports which support Fagan's claims (Ackerman, 1989). However, using inspection does involve the employment and training specialised inspectors.

8.3.3 Testing

Walkthrough and inspection alone are not sufficient to guarantee the total correctness of software product. System test is another step to improve the software quality at the late stages of the software development process. The objective of system testing is not to show how good the software is, but to uncover any possible defects in the software.

There are many different types of software testing. The usual classification includes unit test, integration test, validation test or acceptance test according to which stage it is applied; or functional test, performance test, stress test, structural test, usability test, interface test and so on according to its purposes (Fairley, 1985).

The system testing can also be divided into two groups, bottom-up testing and top-down testing (Bell, 1992; Sommerville, 1989).

- Bottom-up testing starts the test process at the bottom modules in the system hierarchy. It proceeds to the sub-system level and finally to the entire system.
- Top-down testing is to start the testing with the main system routine, then proceeding to sub-system routines and bottom modules.

For example, in a system of calculating building energy consumption, the top routine may be for energy consumption, one sub-system routine is the building fabric heat loss and one unit module could be the calculation of elemental U-value. The bottom-up testing of such a system would start from elemental U-value to building fabric heat loss to building energy consumption. The top-down testing would proceed in reverse order (figure 8.3).

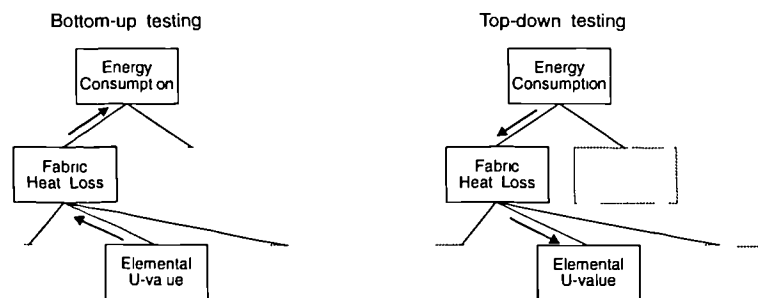


Figure 8.3 Two software testing approaches

Another similar classification is to divide system test into white box testing and black box testing (Pressman, 1987).

- White-box testing is conducted on the base of knowing the internal workings of the software. The objective is to ensure the internal operation performs according to specification and all internal components have been adequately exercised.
- Black-box testing is conducted on the base of knowing the specified functions of the software. The objective of such a test is to demonstrate that each function is fully operational.

The bottom-up and top-down approaches are usually used in combination for different tasks. For unit testing the bottom-up approach is usually used, and for integration testing the top-down approach is more appropriate.

8.4 Testing Strategy for MultiCAD

It has been stated already that MultiCAD is only at the prototype stage with the purpose of testing the concept for an integrated building design system. Therefore, the system testing requirement is not as high as it should be for commercial oriented software products.

During the development process, reviews on the prototype are conducted periodically by the programmers concerned. This section concentrates on the testing and evaluation at the late stage of the implementation and immediate after the prototype implementation.

8.4.1 Objectives and tasks

At this stage, the emphasis of the testing is on the unit test and integration test. The objective is to ensure the internal integrity of the prototype and the compliance with the specification set out in section 6.6. The validation test itself is beyond the scope of the prototyping process.

8.4.2 Test plan

Based on the above objectives, the testing of MultiCAD prototype covered the following three stages.

1. Unit test.
2. Integration test (stage 1).
3. Integration test (stage 2).

Unit test is to ensure the basic components of the software are correct. Since MultiCAD is implemented in an object oriented paradigm, the task of unit test is to guarantee the correctness of all the methods of each object. Integration testing (stage 1) is to ensure each functional module, i.e., energy calculation, daylight, etc., performs correctly. Integration test (stage 2) is to ensure the prototype, as a whole, satisfies its specification. The testing procedure is indicated by figure 8.4.

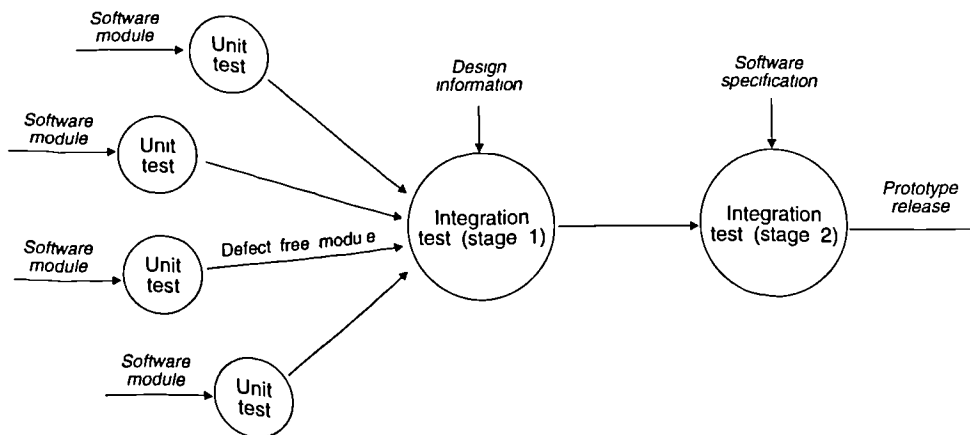


Figure 8.4 MultiCAD testing procedure

8.4.3 Unit testing

The objective unit testing is to ensure the correctness, robustness as well as efficiency of all the components of the software. It includes functional test, performance test, stress test and structural test. Fairley (1985) explained the objective of each activity.

- *Functional test* involves executing the code with nominal input value for which the expected results are known.
- *Performance test* measures the speed of the code execution.
- *Stress test* is used to determine the strengths and limitations of the software by intentionally breaking the program units.
- *Structural test* is concerned with the internal logic of a program.

During the unit testing of MultiCAD, the following tasks were conducted, *Functional test*, *Performance test* and *Structural test*. *Stress test* is addressed in the integration test stage 1, because it is much easier to test the system with various combinations of input and user actions using the MultiCAD interface.

Since the object oriented paradigm associates the functions with the objects, the unit testing is a matter of creating objects and testing their functional behaviour or methods.

If errors are found during software testing, the next task is to locate and correct them. This process is usually known as software debugging. Actor, the programming environment in which MultiCAD is implemented, has a very powerful debugging tool. It also includes several built-in checking mechanisms in order to improve the code quality.

The following is a list of tools used during the MultiCAD unit test process.

- *MultiCAD interface* is used to create objects, i.e., a building, a room, a wall, etc.
- *Actor Inspector* is used to inspect the attributes of an object. It is also used to issue testing commands to the object concerned.
- *Actor Browser* is used to browse the object method list as well as the source code. It performs a syntax check during the interactive code compiling. Any irregularity is prompted for correction.
- *Actor Debugger* comes up automatically if any error is detected during code execution.
- *Actor Profile* provides useful statistics on program execution. It shows the methods being used during a particular execution routine as well as the number of times they have been used.

Figure 8.5 shows a flow chart of the unit testing procedure of MultiCAD.

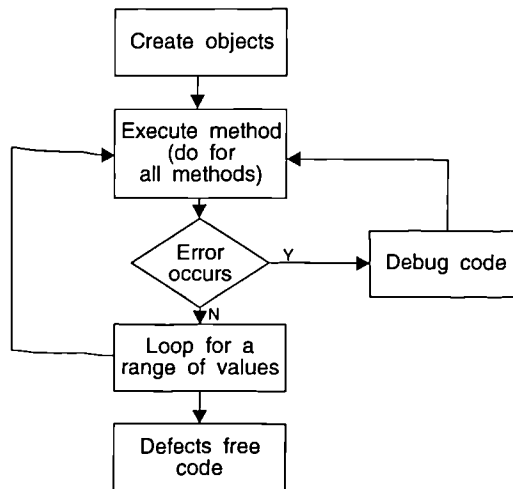


Figure 8.5 The procedure of unit testing

- (1) The starting point of the unit test is to create objects, e.g., buildings, rooms, walls, roofs, windows, layers, finishes and so on. All the object classes define method for creating new instances normally called *new(self)* or *defaultNew(self)*. Therefore, creating new object is merely a matter of executing the method. The MultiCAD interface defines the routine of calling the object creating methods and instantiate object attributes with default values. It is used as a convenient approach for creating objects.
- (2) Once an object is created, it can be inspected using the Actor Inspector. All the attribute values can be displayed and object methods can be executed in the editing area of the Inspector.
- (3) If any error occurs during the execution of a method, the Debugger comes up and shows where the error is located as well as information indicating the possible causes of the error.
- (4) After the correction, the execution can be continued. This debugging process is repeated until the method execution routine goes through without problem.

This testing is conducted from the lower order to higher objects, or from objects such as wall, to room, then building. This is consistent with the bottom-up approach and the following example is presented to illustrate the conduct of the unit testing of MultiCAD.

A simple building with two rooms is created using the CAD tool of the Building Editor (figure 8.6). The data of the building is instantiated as it is created, e.g., its geometry, elemental construction, etc. These default values can be changed by the user. For the purpose of unit testing, the default values are perfectly valid.

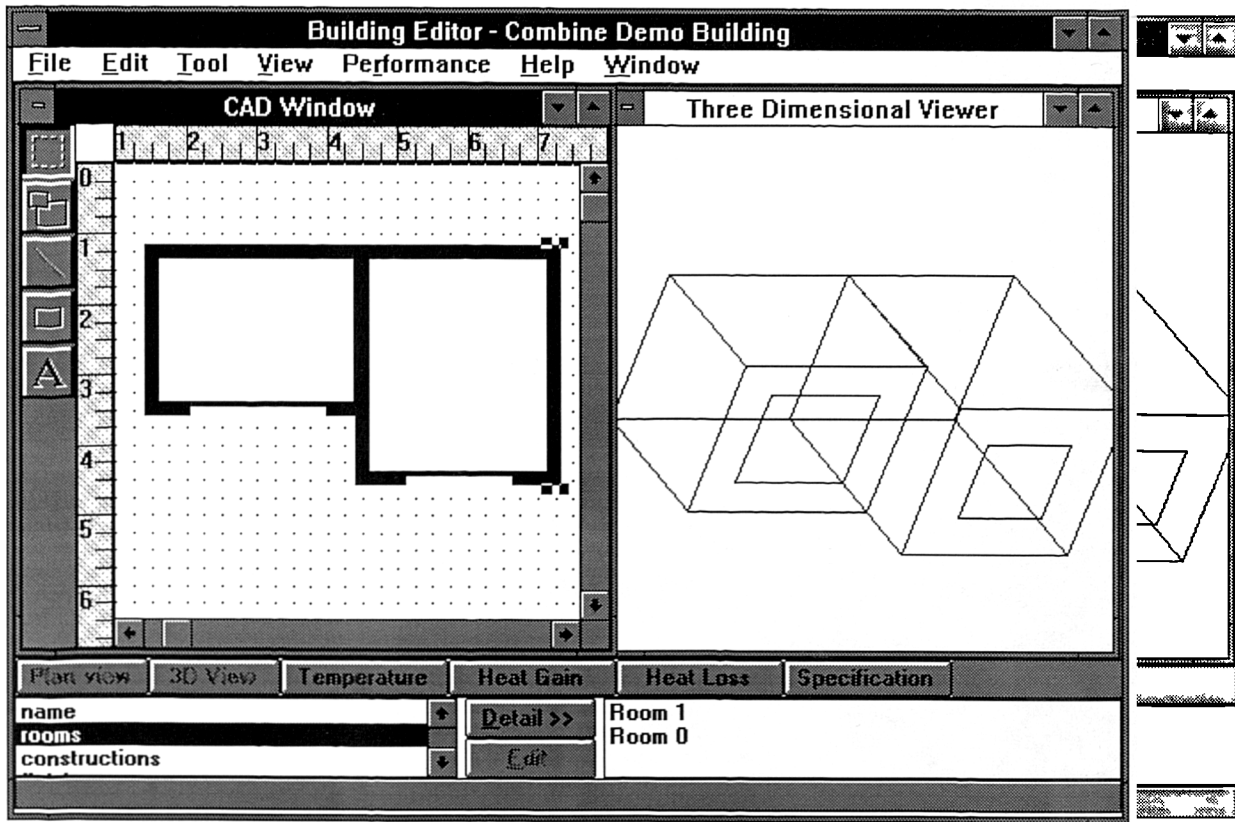


Figure 8.6 Creating objects using MultiCAD interface

When the building is created using MultiCAD, the result is not a single building object, but a complex building system with many objects and their inter-relationship. In this example, there are two rooms, room 0 and room 1. Each room is enclosed by four walls, a roof, a floor and a window. All walls have specified geometry and construction. The wall construction is made of a number of layers. Each layer is made of a particular material with certain thickness. All these components of the building system, e.g., building, room, wall, windows, layer, etc., are objects. The framework in which they exist is the underlying data model of MultiCAD.

Once an object has been created, it can be inspected using the Actor Inspector (figure 8.7). In this example, the object being inspected is the wall which is marked in figure 8.6.

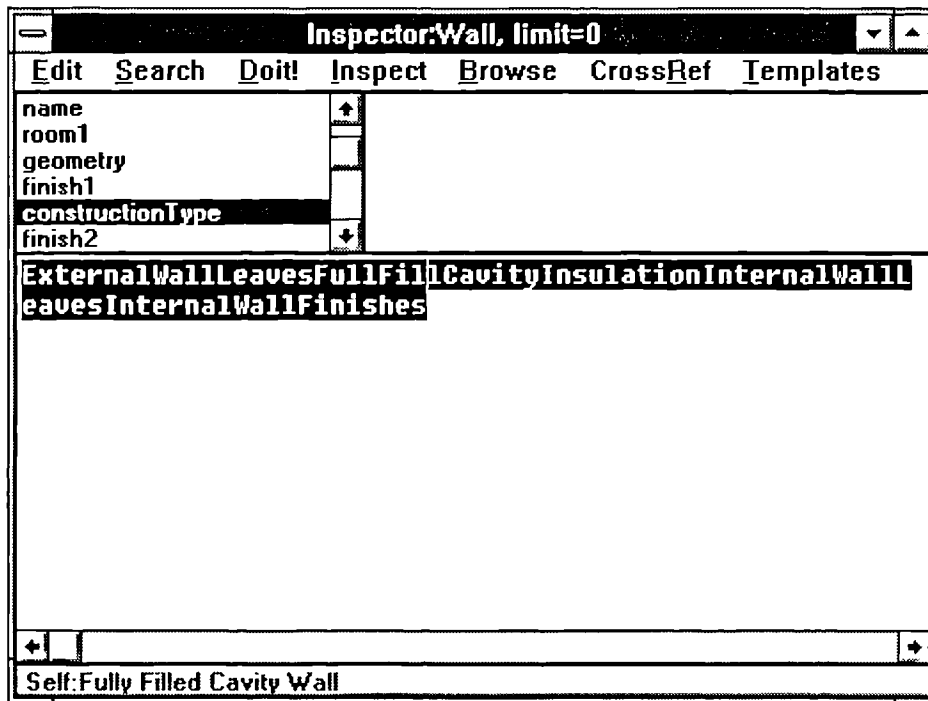


Figure 8.7 Inspecting object

The object Inspector is able to display all the instance variables or the attribute items of the object, in this example the name of the wall, its connected rooms, geometry, etc. In addition, the Inspector can be used to examine its behaviour by executing its methods in the editing area.

The Actor Browser is used to get access to object methods, their name and the source code (figure 8.8).

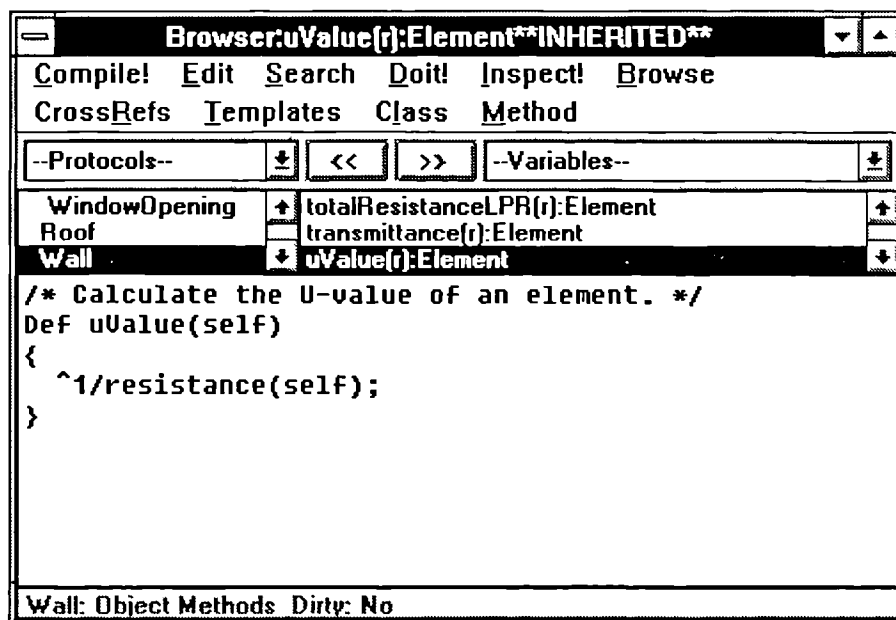


Figure 8.8 Browsing the object methods

This particular example also demonstrates the inheritance mechanism of the object oriented programming. Since Wall, Roof, etc., are descendants of Element, they inherit the method defined in the Element class, the *uValue* method. The *uValue* method is defined as *1/resistance(self)*. *Resistance* is another method defined separately by each descendant, in the case of wall it is defined as follows:

```

/* Calculate the total resistance of a construction including surface resistance.
*/
Def resistance(self)
{ ^surfaceResistance(self,room1)+totalResistance(constructionType)+
  surfaceResistance(self,room2);
}

```

In the *resistance* method, several other methods are called either to Wall itself *surfaceResistance(self)* or to one of its instance variables which is another object, *totalResistance(constructionType)*. All these methods can be browsed in the Browser.

Methods in object oriented programming are equivalent to functional modules in traditional procedural programming. Therefore, the unit test in OO program is to test the object methods.

Testing the *uValue* method for a wall, it is done simply by sending the command to the wall being inspected (figure 8.9) in the Inspector.

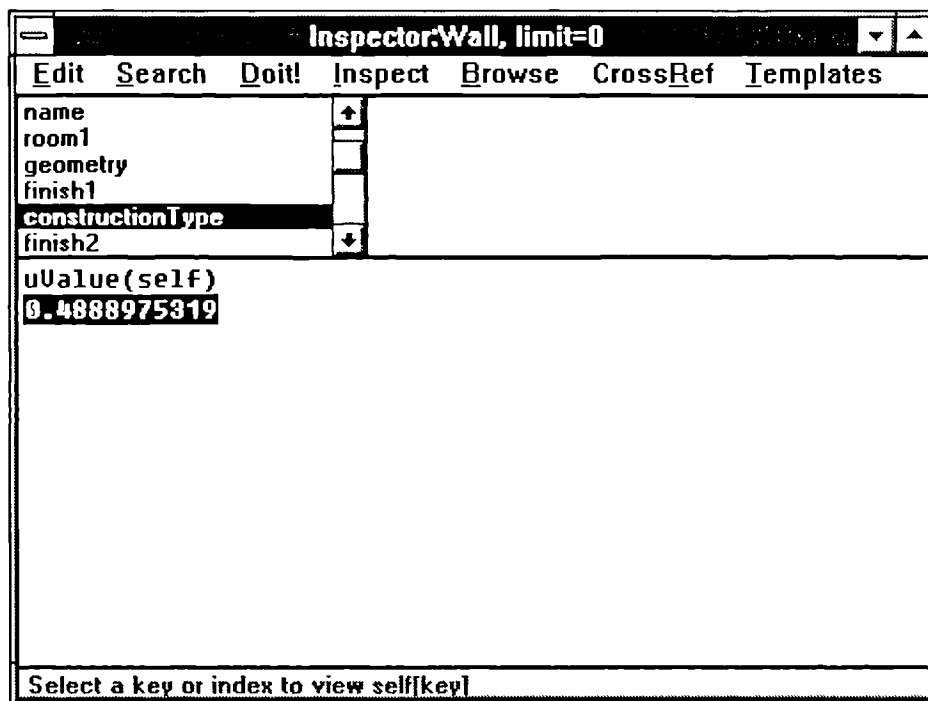


Figure 8.9 Executing object method

In this case the execution does not encounter any error, it produces a result of 0.4888975319 for the U-value of this wall. For the purpose of demonstrating the debugging procedure of Actor programming environment, an error is intentionally introduced into the U value calculation routine. When the code is re-executed a debug message box comes up (figure 8.10).

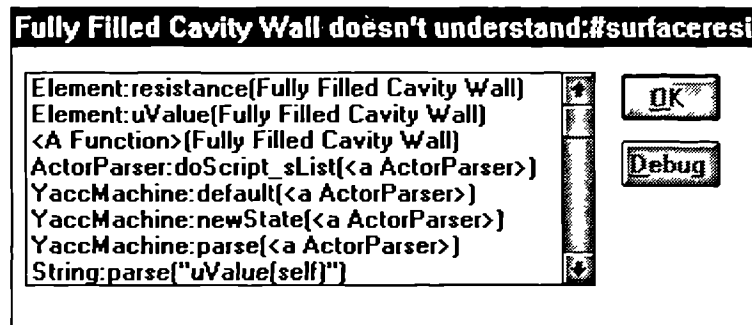


Figure 8.10 Debugger prompt is shown when an error is detected

This prompt message indicates an error is detected during the execution process. The list in the box shows the sequence of the execution. The one at the top is usually the one where the error occurs. It is possible to go to the source code from the debugger prompt by either click the Debug button or double click on a line in the list box, preferably the top line since that is where the error might occur (figure 8.11).

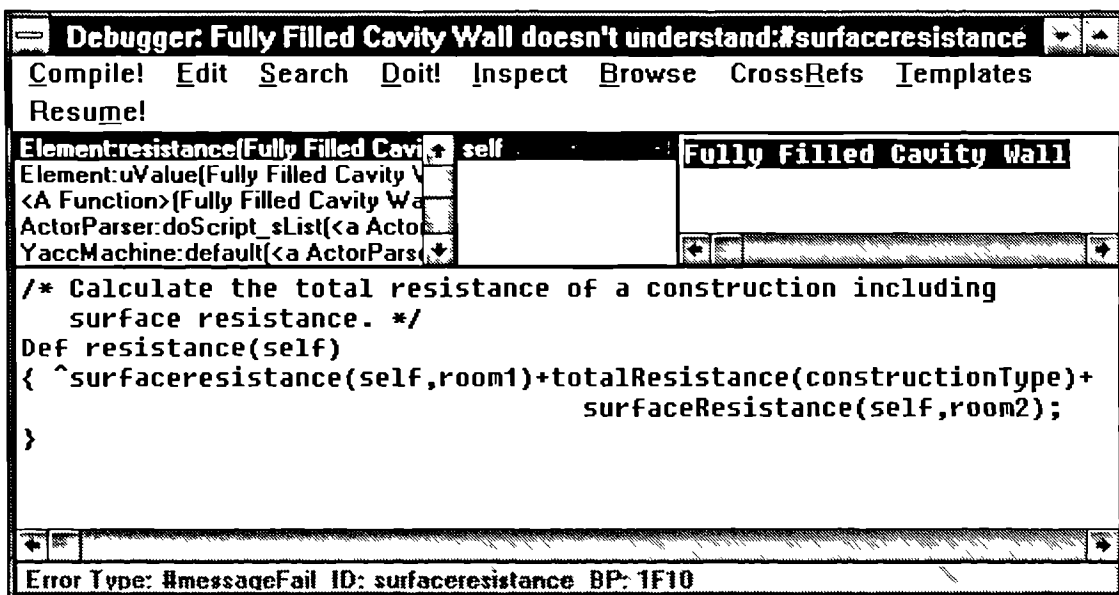


Figure 8.11 Actor Debugger for program debugging

The error is indicated on both the title bar and the help bar at the bottom of the debugger window. In this case, on the title bar, it reads "Debugger: Fully Filled Cavity Wall doesn't understand:#surfaceresistance". On the help bar it says "Error Type: #messageFail ID: surfaceresistance BP: 1F10". Based on these evidences, it is not difficult to conclude that the error is the method *surfaceresistance(self,room1)*. Further checking with the method list for Wall class shows that there is no such a *surfaceresistance* method. The one it has is *surfaceResistance*. Therefore, clearly it is a typo in which the capital 'R' is mistaken by lower case 'r'. After correcting this error, the execution can be resumed by choosing the *Resume* option on the Debugger menu.

The unit test is conducted, using the procedure described in the above example, to all the methods of each object class of the data model. Apart from the type of error shown in this example, speed performance evaluation is carried out to some of the time consuming methods. Actor provides a *StopWatch* class which can be used to measure the speed of the program operation. Its usage is as follows:

```
start(Sam:=new(StopWatch))
hourlyEnvironmentalTemperature(aRoom,12)
stop(Sam)
```

The first line is the command to start the watch, second line is the method under investigation and the third line is to stop the watch after the execution is finished. Performing the above three commands in succession produces the time spent on the execution in milliseconds.

13787L

In this example, the time spent is 13.787 second. If in some cases the execution takes suspiciously long time, a detailed investigation is possible using the Actor profile mechanism.

```
profile()
hourlyEnvironmentalTemperature(aRoom,12)
profile()
```

The profile command using as shown above produces a text file, *profile.dot* which records all methods being called as well as the number of times they are called. The format of each record is [method name:class name->times of use]. The following is the file produced for the execution of the method *hourlyEnvironmentalTemperature(aRoom,12)* for a room.

```
makeLiteral(r,p1):ActorAnalyzer->1
meanGain(r):Room->1
<A Function>->1
firstExternalElement(r):Room->1
compile(r):Object->1
doObj_msg(r):ActorParser->1
```

```

totalInternalAdmittance(r):Room->1
doObj_sys(r):ActorParser->1
last(r):Stream->1

...

init(r):Schedule->2
initCasualSchedule(r):Room->2
perimeter(r):GroundFloorSolid->2
admittanceMatrix(r):Roof->2
dayByHour(r,p1):ScheduleClass->2
uValueAnderson(r):GroundFloorSolid->2
getIdent(r):ActorAnalyzer->2
doScript_sList(r):ActorParser->2
initResistance(r,p1):AdmittanceMatrix->2
abs(r):Real->3

...

find(r,p1):Array->245
in(r,p1):Array->245
hourlySolAirTemp(r,p1):Element->269
dec(r):Number->294
=(r,p1):Number->304
do(r,p1):OrderedCollection->326
?hasElements(r):OrderedCollection->356
**(r,p1):Number->380
init(r):OrderedCollection->429
setImaginary(r,p1):Complex->472
over(r,p1,p2):IntervalClass->525
do(r,p1):Int->525
new(r,p1):CollectionClass->603
setReal(r,p1):Complex->632
new(r):ComplexClass->632
init(r):Complex->632
ancestors(r):Behavior->784
new(r,p1,p2,p3):IntervalClass->851
/(r,p1):Number->935
-(r,p1):Number->1338
add(r,p1):OrderedCollection->1380
*(r,p1):Number->1568
+(r,p1):Number->1830
asReal(r):Int->2391
converterFor(r):Real->2833
generality(r):Real->2917
converterFor(r):Long->3528
generality(r):Long->4288
generality(r):Int->5905
coerce(r,p1,p2):Number->6433

```

After getting the profile information, two main measures can be taken to improve the performance of the program. (1) Rearranging the program to eliminate unnecessary message passing. (2) Optimising the methods which are used heavily. In one example, the time for calculating hourly temperature in a day for a room was reduced from more than five minutes to less than one minute.

There is another type of error which is more difficult to debug. Sometimes an execution runs through without any problem but the result is incorrect. There are a number of possible causes for this error. It could be caused by the wrong algorithms being used. It could be caused by mistyping of values in the source code. It could also be caused by unintentional violation the programming language conventions. Detecting and locating these errors often needs manual checking and logical analysis. The following are several examples of this kind.

- (1) Once the calculation of room volume was consistently 10 times bigger than it should have been. The error responsible for this is a '0' is missing at one of the number in a method of this calculation routine.
- (2) In the calculation of the hourly room temperature, when a window is added to the east facing wall, one expects during the summertime the room temperature should increase in the morning. When the result shows the temperature increases in the afternoon instead of morning, it can be assumed that errors must exist in the calculation routine. Debugging finds that it is the orientation method for windows was wrong. Instead of returning 90 degree which is east facing, the orientation method returns 270 degree which implied a west facing window.
- (3) Initially, the ventilation heat loss method for rooms was implemented as follows by mistake:

(1/3)*volume(self)*airChange

This method appears to be perfect valid. However, in Actor the operation for Integer 1/3 results in '0' instead of '0.3333' and this method always returns '0' instead of the expect value.

The unit testing for MultiCAD described above ensures its underlying functional integrity. However, it has to be acknowledged that system testing is a lengthy process and no software can claim absolutely error free. The objective of the testing is to reduce errors to an acceptable level.

8.4.4 Integration testing (stage 1)

The objective of integration testing is to ensure the software performs well not only as individual modules but as a whole. In fact, the bottom-up unit testing has performed the integration testing to certain extent. When a complex method for a complex object is executed, i.e., the *hourlyEnvironmentalTemperature* method for a room, it is effectively an execution of an integrated programming routine. In the example of room *hourlyEnvironmentalTemperature*, many other methods or smaller modules are called, such as *uValue* for elements, *surfaceArea* for walls, *meanSolarRadiation* for windows and so on.

Apart from the object classes in the data model, there are many interface classes in MultiCAD application. The integration testing of MultiCAD tests the integration of these two parts of the program as well as the stress test using boundary and beyond boundary input values. At stage 1, each functional module, such as energy, daylight, heatloss, is tested separately.

One of the examples of the unit testing was the U value calculation of a wall. The command was then issued in the editing area of the Actor Inspector. The final user interface for the U-value module is shown in figure 8.12.

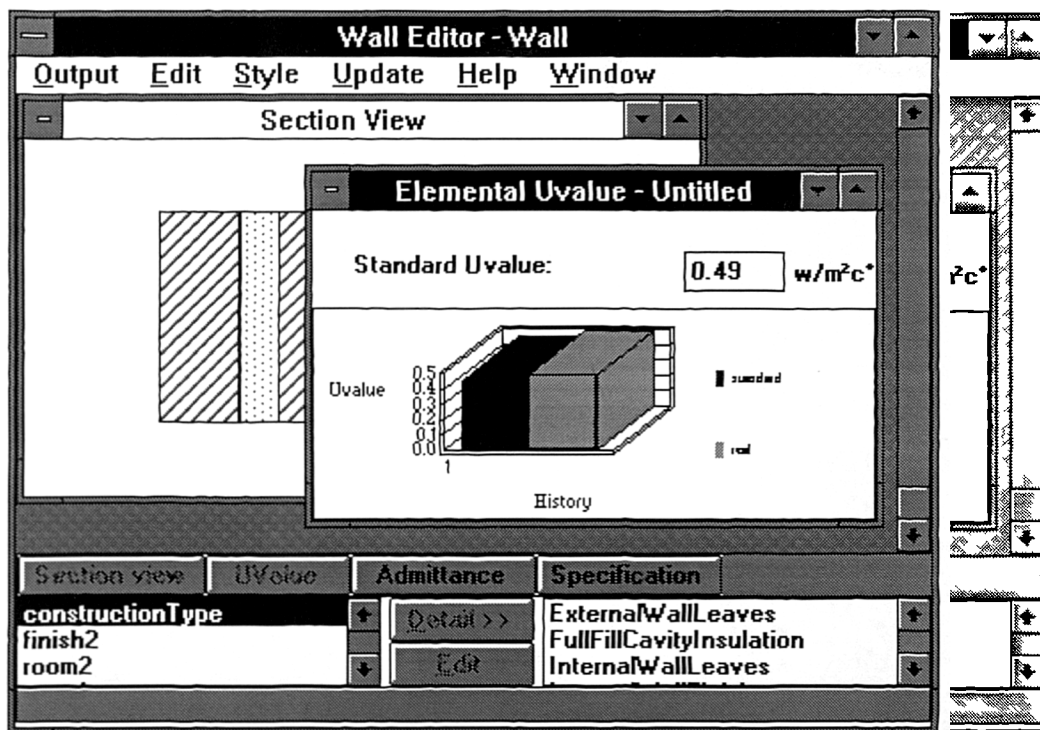


Figure 8.12 Interface for the elemental U-value

The calculation result is shown in a box on top, and at the bottom part is a graphic history of the U-value change as a result of the change in element construction. One of the changes is the editing of a particular layer using the MultiCAD Layer Editor (figure 8.13).

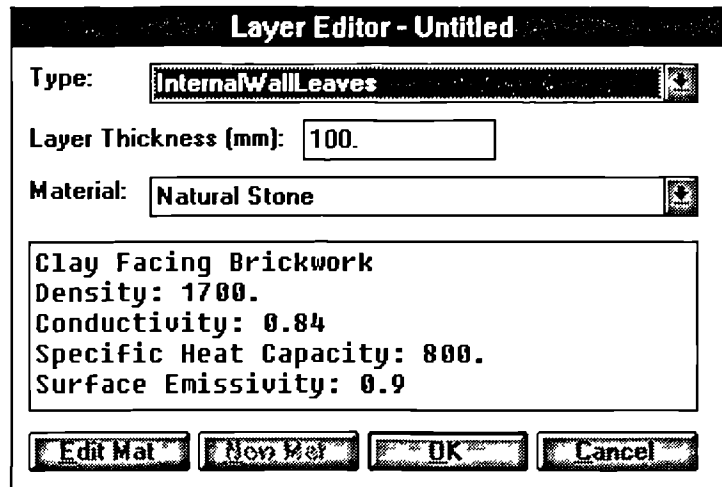


Figure 8.13 MultiCAD Layer editing interface

As expected, changing the layer specification results the change of the wall specification. Consequently, a new U-value is calculated for the wall. However, this does not exclude possible system failures under certain unexpected circumstances. For example, in the Layer Editor, if an extreme number, i.e., '100000' is typed in for the layer thickness input it causes an error (figure 8.14).

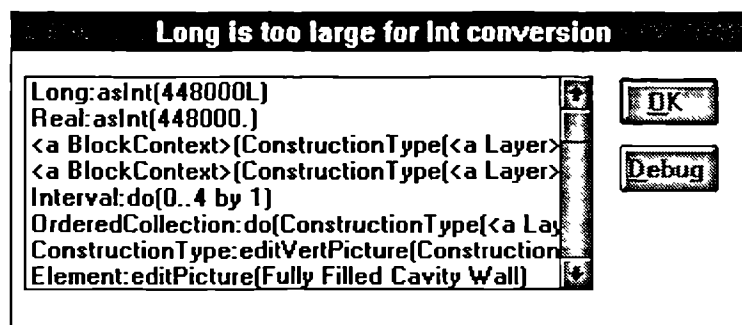


Figure 8.14 An example of stress testing errors

Although this type of error is not common, it could nevertheless happen. Therefore measures are needed to prevent their occurrence. In this case an input range checking mechanism is introduced into the program. Instead of a crash, the program puts up an error message to warn the user and prompt a correction, and then the program continues the operation (figure 8.15). Measures of this kind enhance the integrity and robustness of the software.



Figure 8.15 An example of additional safety checking

During this stage of integration testing of MultiCAD, various combinations are tested for all major functional modules of the system.

8.4.5 Integration testing (stage 2)

The integration testing (stage 2) of MultiCAD concentrates on the evaluation of the program against the system specification setting out in section 6.6. The following issues are identified as the primary testing tasks for this stage.

- To ensure MultiCAD supports multiple design tasks.
- To ensure MultiCAD supports interactive information exchange between different tasks.
- To ensure MultiCAD supports three of the four types of design requests for each domain identified in section 6.5. The fourth type of request, calculating the value of a design parameter to meet a required design target, has only been implemented for a daylight design tool (section 7.10).

Multiple design tasks support:

MultiCAD, as specified in section 6.6, is able to support multiple design tasks. Once a building object is created using the CAD interface the following tasks can be performed:

For the building,

- annually energy consumption,
- monthly heat losses,
- monthly heat gains,
- monthly temperatures.

For rooms of the building,

- average daylight factor,
- hourly temperature,
- heat load.

For elements of the building (walls, roofs)

- U-value,
- admittance,

Because, the information about the building is held in the central data model, the user does not have to provide input for each operation. Therefore, the tools are much easier to be used. Figure 8.16 shows an example of a Building Editor which includes a CAD tool through which a simple building of five rooms is created. A number of evaluations have been performed on the building, i.e., monthly heat losses, monthly heat gains, monthly temperature. The results are shown graphically in separate windows.

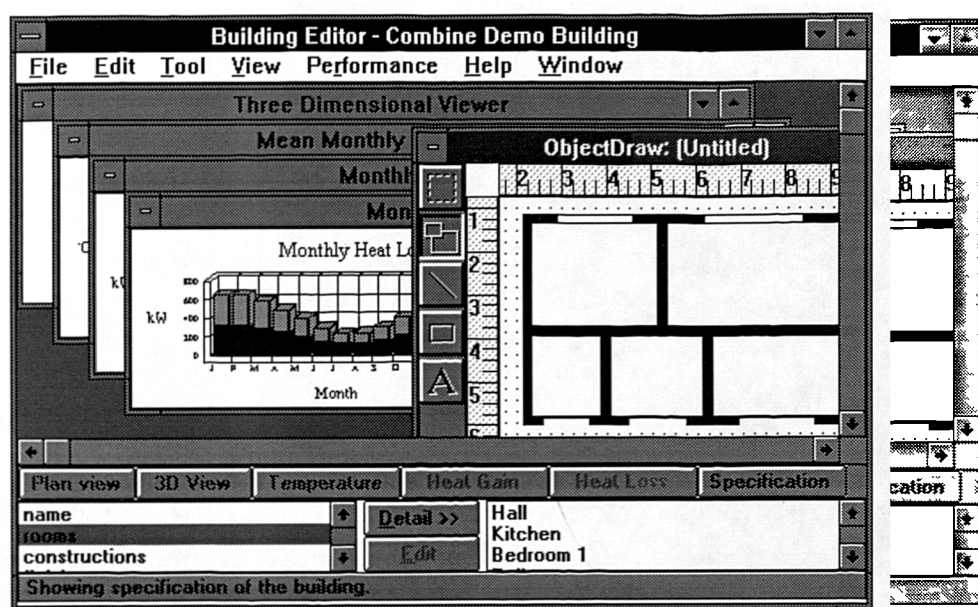


Figure 8.16 Multiple design tasks support 1

The user may decide to do some detailed evaluations on parts of the building, i.e., a room. Doing so, the user only needs to identify the room in the CAD window or in the property list box and selects the edit button, a Room Editor is opened on the screen and daylight, hourly temperature and thermal load calculation can be carried out (figure 8.17).

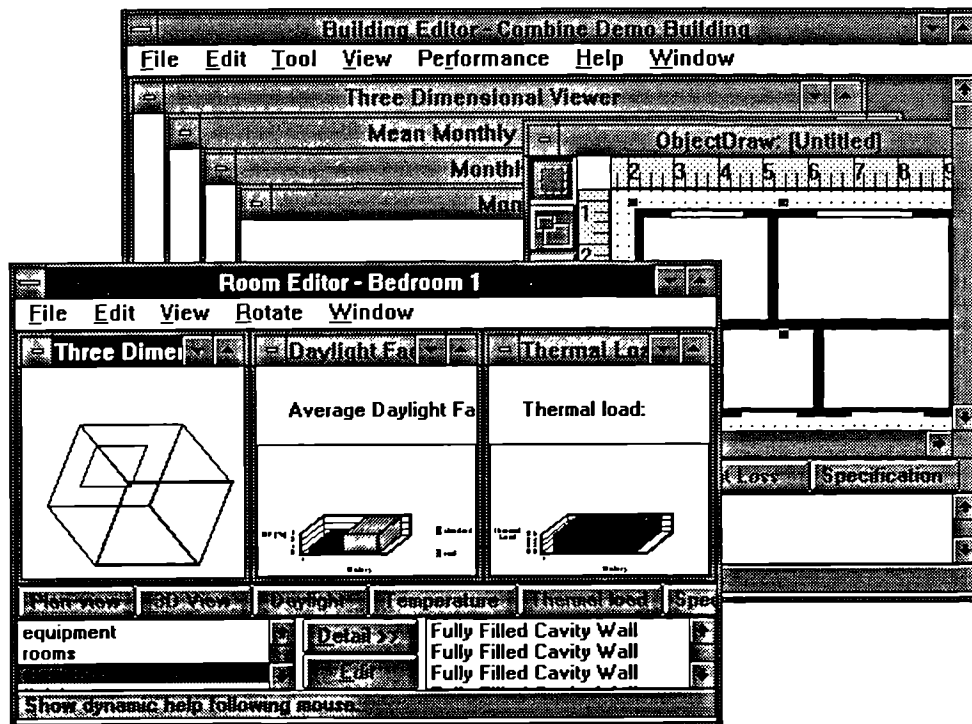


Figure 8.17 Multiple design tasks support 2

Similarly, an element can be identified, from either the Room Editor or the Building Editor, for U-value calculation (figure 8.18).

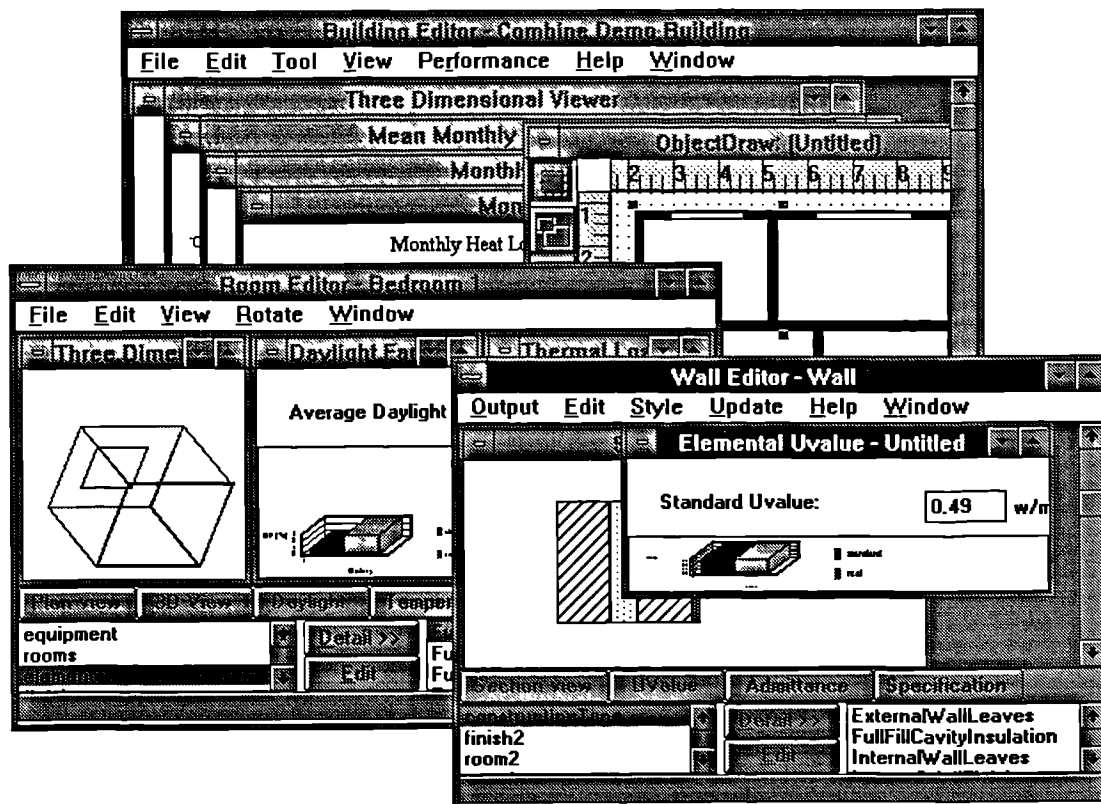


Figure 8.18 Multiple design tasks support 3

Interactive information exchange:

Since all the design tools are integrated through the data model, data exchange between them is handled automatically by the system. Design changes made in one tool is reflected dynamically in other relevant tools. The following is an example of the data exchange at work.

Figure 8.19 shows a room editor with a graphic display of daylight and heat load calculation results. In the same figure, there is a wall editor operating on a particular wall of the room. The U-value calculation result is shown graphically.

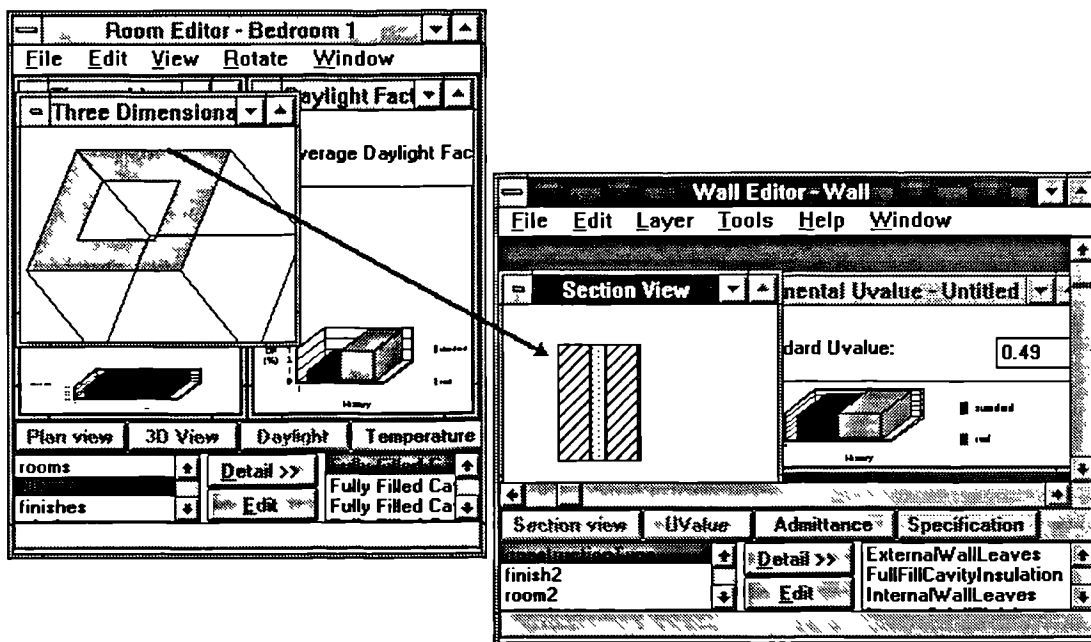


Figure 8.19 Data exchange demonstration 1

The next step in this example is to change the construction of the wall from 'fully filled cavity wall' to a 'solid wall' (figure 8.20).

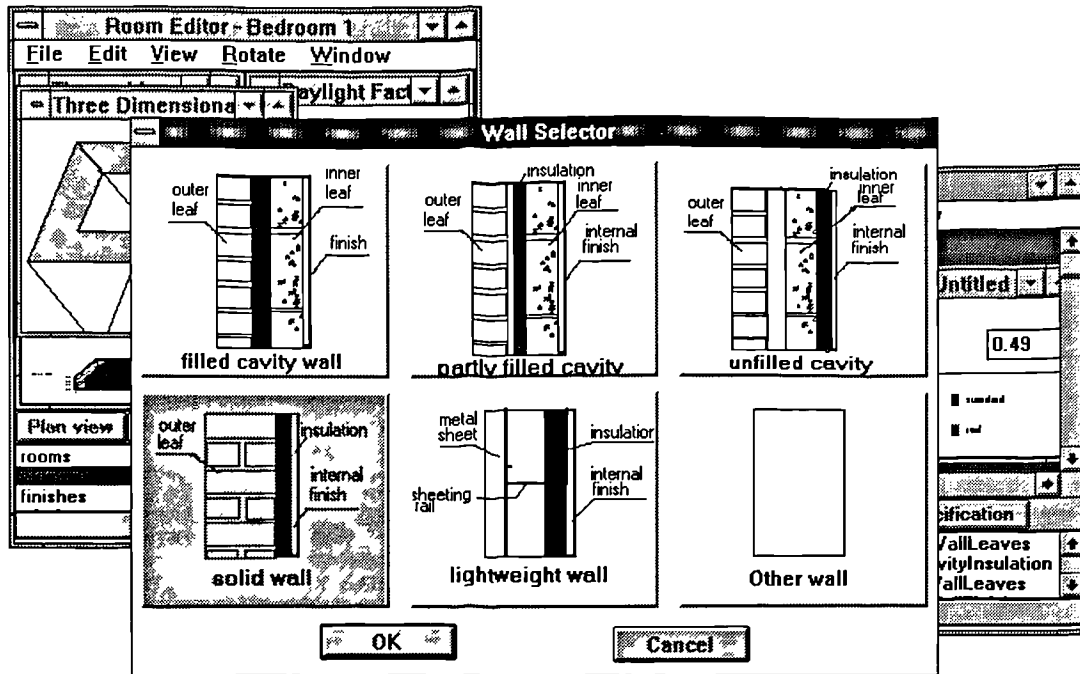


Figure 8.20 Data exchange demonstration 2

As a result of the construction change of the wall, its U value has been re-calculated and the result is displayed in the graph (figure 8.21). At the same time, the change is sent to the underlying data model which in turn informs the room editor that the construction of one of its wall has changed. The room editor re-calculates the daylight factor and heat load for the room and displays the result as shown in the figure 8.21.

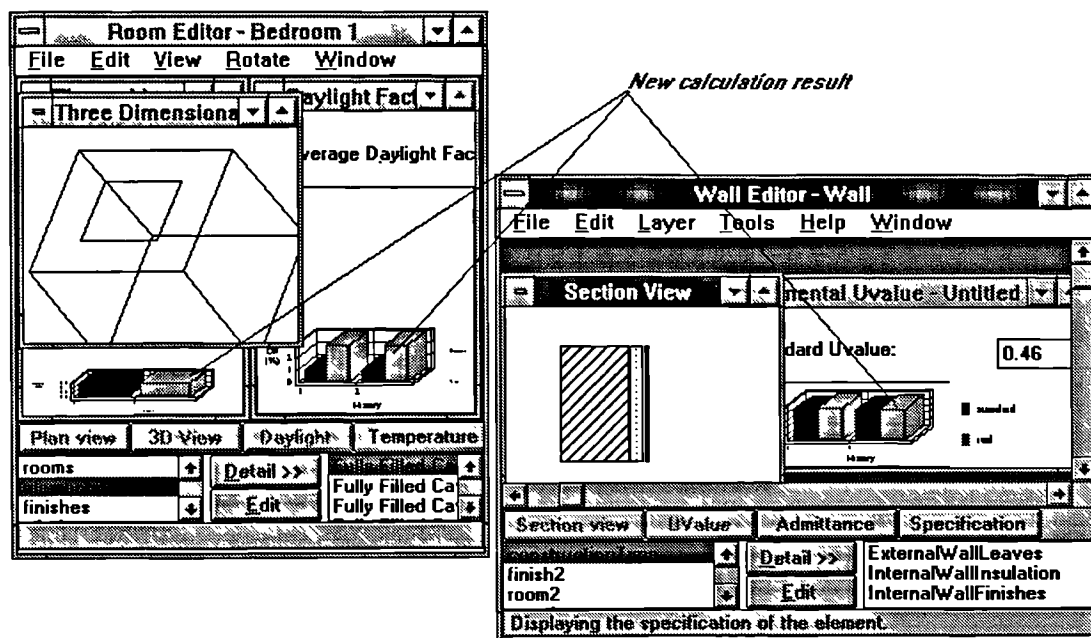


Figure 8.21 Data exchange demonstration 3

This kind of interactive data exchange is maintained for all the tools. When several design tools are used simultaneously, a minor data change could trigger a lengthy re-calculation for all the tools. To prevent this from happening, an update lock mechanism is introduced for each tool, especially for those requiring longer calculation. Using the update lock, the user is able to control whether an interactive update is desired for each individual design tool.

Three types of design requests:

MultiCAD is able to carry out straight forward calculation in building energy, room daylight, room overheating, elemental U-value, etc. (figures 8.16, 8.17, 8.18). This fulfils the second type of design request set out at section 6.5, which is the 'evaluation of a design solution against specific criteria'. In cases when an authoritative standard is available, i.e., CIBSE daylight and Building Regulation U-value requirement, the standard is displayed along with the calculation result. The user can very easily tell whether his design meets the standard requirement.

The third type of request is 'sensitivity test, which is to examine the impact of a parameter(s) on the result'. This is demonstrated in the above data exchange example (figures 8.19, 8.20, 8.21). In that example, when the construction of the wall is changed, its impact on the u-value is shown. Changes, such as adding a new layer, removing an existing layer, changing the specification of a layer and so on, all influence the elemental u-value. MultiCAD is able to show the impact of these changes by displaying not only numeric calculation results but also a graphical change history. The user can see which change makes the result better or worse. Similarly, energy, daylight and overheating are all able to respond to changes in design parameters. All the design parameters can be changed through user interface described in section 7.7.

Another type of design request is to provide 'general guidance on a design aspect'. MultiCAD utilises the help mechanism provided by the Microsoft Windows (Microsoft, 1990). The help system is able to do keyword searching, context jumping, sequential browsing, etc. It is an ideal vehicle to provide well-compiled design information support. Figure 8.22 shows the help system for daylight application, whose content is based on the CIBSE window design guide (CIBSE, 1987). In addition, MultiCAD provides help on U value calculation and Part L of the Building Regulations.

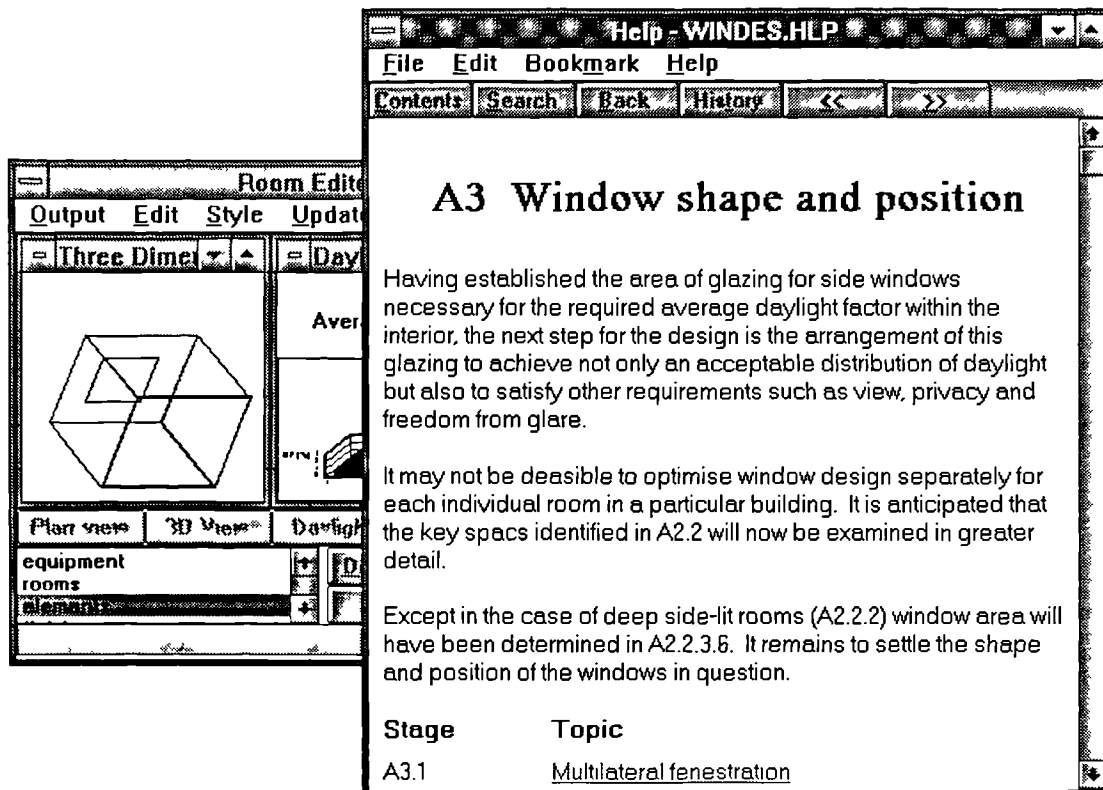


Figure 8.22 MultiCAD supplies general design information support

The integration test (stage 2) has demonstrated that MultiCAD meets its functional specification and design requirements. The included examples only show a small part of the software and the tests that have been conducted. Nevertheless, they illustrate the successful implementation of the prototype of an integrated building design system. MultiCAD provides an integrated design support environment. It would allow an architect to perform multiple design tasks on the same building object. It provides automatic data exchange between different design tasks and supports several types of design request. Systems of this type has the potential to provide architects with technical support which might otherwise not be available or difficult to get as argued in the first part of this study. Therefore, it is fair to say that the development of MultiCAD is a step forward in the improvement of technology transfer process in the building industry.

8.5 Summary

Verification and Validation are activities aimed to building quality into a software product. The common V&V measures include review, walkthrough and testing. They are carried out at each phase of the software development process.

This chapter focuses on the testing and evaluation after coding. Since MultiCAD is a prototyping stage, the testing effort is concentrated on unit testing and integration testing.

The objective is to ensure the prototype is relatively stable and performing its functions according to the specification.

CHAPTER 9

CONCLUSIONS AND ISSUES FOR FUTURE STUDY

The final chapter is organised into two parts. The first part is a review of this study. The second part discusses several issues raised during the development of MultiCAD and makes some suggestions for future research in this area.

9.1 Review of This Study

Technology transfer in the building industry, or the lack of it, has been a cause of wide spread concern in the research community. The main objective of this study has been to examine this issue in order to identify obstacles and to propose ways to improve the process. Progress has been made in addressing this complex subject and the major issues which have been addressed are summarised below.

9.1.1 Overview of the technology transfer in the building industry

Research and practice in the building industry have become very specialised and independent disciplines during the past few decades (section 2.3). Most researchers in the academic research community concentrate on the investigation of building performance, but they do not get directly involved in building practice. Therefore, their research achievements have to be transferred into practical application in order to realise their potential benefits to mankind. Unfortunately, it has been widely acknowledged that at present this technology transfer process is not effective due to various reasons one of which is the communication gap between the research community and practitioners (figure 2.6).

Building practice involves many parties, owners, developers, architects, planners, surveyors, contractors, builders, manufacturers, legislators and so on (BRE, 1992). All these parties have various degrees of influence on building design and construction. However, it has been argued in this study that architects have a key role to play in the technology transfer process, because they usually make the major decisions during the design stage, and these decisions often have the most influence on the final performance of the building. Therefore, they are the targeted party of this study (section 2.5).

9.1.2 Identification of barriers of the technology transfer process

Technology transfer requires that architects are provided with access to existing knowledge to enable them to make design decisions on a well-informed basis. In order to provide this help to them, it is necessary to understand several important issues concerning their method of working, such as how do they carry out design? What kind of technical support do they need? How should this help be presented?

Architectural design is an extremely complex process. Despite several studies in recent years, there is no commonly accepted model to describe the design process. Nevertheless, some conclusions have been drawn by various researchers.

There are two decision making strategies, problem-oriented and solution-oriented. Architects adopting the first strategy usually use an Analysis-Synthesis-Evaluation pattern of design method. On the other hand, architects adopting the latter design strategy often follow a design pattern of Generator-Conjecture-Analysis. In other words, these architects normally generate design solutions quickly when confronting a design problem, then check the fitness between their solutions and the design conditions. The design solution is modified again and again until the architect believes that it sufficiently satisfies the design requirements. A design problem is usually very complex, and it is difficult to apply exclusively either a problem-oriented design method or a solution-oriented method in practice. More often than not, architects adopt, consciously or unconsciously, a combination of these two approaches (section 3.3.2).

No matter which design approach the architect adopts, generating design ideas and evaluating them are the two essential activities during the design process. On the basis of the requirement of these two activities, design supports can be classified as generative and evaluative (section 3.5.2). Generative design support can be considered to include design advice, guidelines, and other general information while evaluative design support covers algorithms, tables, monograms, computer calculation programs and so on.

While in the long term design support should be provided through out the whole design process, in order to make progress in the short term a specific domain has to be addressed in any individual research project. This study takes the early design stage and energy oriented issues as the area of interest. The reason for this is that early design stage is crucial to the performance of the final building, and at present the design support for this stage is the weakest (section 3.4).

Several types of design support have been identified, exemplars, rules of thumb, evaluation tools, trade literature (section 3.5.2), and importantly, there are some criteria which they must meet to gain acceptance. They must be valid and authoritative, interactive and responsive, accessible and design oriented (section 3.5.3).

Unfortunately, as discussed in chapter 4, many existing architectural design support systems do not satisfy the above criteria. The common deficiencies can be summarised as follows (section 4.4):

- Many design guides and handbooks, i.e. those for passive solar design, do not have the authority needed to encourage design professionals to apply them in practice.

- Printed material, the most commonly used means for technology transfer in the building industry, is not very effective for interactive communication.
- Many existing design support tools concentrate on a single and narrow facet of the building design, e.g., daylight, heat loss, etc. They do not fit the nature of architectural design which is a multi-tasking and interactive process.
- One problem with the discrete provision of design support tools is that the scattered information sources are not always available to individual designers. In addition, especially in the case of computer applications, the lack of standardisation means that each tool has its own way of building representation, data format and operating style. The integration between different tools is almost impossible.

The authority issue underlying knowledge for design support tool is important but not the direct concern of this study. The focus is on how to transfer the existing valid knowledge to designers.

The increasing penetration of computers in architectural design practice during the past two decades has opened new dimensions for the improvement of technology transfer (Chapter 5). They offer several benefits, i.e., massive information storage, fast data processing, accuracy, graphical capability and so on. These technology developments provide the possibility for the creation of Integrated Building Design Systems (IBDS) which include several tools, responsive to architects' design requests and consequently provide help to designers more effectively.

9.1.3 Development of an IBDS prototype

Architectural design is a multi-tasking process. It is not a one dimensional sequential process, but a three dimensional cyclic iterative process. These three dimensions are design phase, design domain and design aspects (figure 6.3). Architects being at the centre of design usually have to consider many factors interactively. This requires that the design support system should be able to handle concurrently multiple tasks.

The requirements set out for the MultiCAD prototype reflect the conclusions of the analysis of the architect needs.

- It must embody multiple design support tools which are valid and authoritative.
- It must be responsive and must have a user friendly interface.
- There must be a dynamic information exchange between different tools.
- The system should be open for future extension.

MultiCAD has been developed in the context of the COMBINE project (section 6.3). The objective of MultiCAD, and indeed COMBINE as a whole, is to prove the concepts of an

IBDS through several prototypes. The domain of MultiCAD is the specification of external building elements at the early design stages. The major consideration is energy related issues, building energy consumption, room daylight, overheating, compliance with Building Regulations Part L and elemental heat loss. The tools selected in MultiCAD are authoritative and widely accepted by the building industry (section 6.7). They are BREDEM-8, CIBSE average daylight calculation, CIBSE summer time temperature, CIBSE standard U-value calculation and procedure described in the approved document of Building Regulations Part L for England and Wales (section 6.7).

In order to ensure the system meets architect's needs, a design request analysis has been conducted and four generic request types have been identified as a result (section 6.5). The system functional specification for MultiCAD has been generated based on the design request analysis (section 6.6).

Data modelling is one of the important issues in the development of an IBDS. The objective of building data modelling is to establish a generic description of the building independent from any particular application (section 5.4.2). The data model can be used as a bridge for the data exchange between all applications.

NIAM and IDEF0 are two formal analysis techniques for data modelling and process modelling. They are used during the analysis stage of the MultiCAD prototype (section 6.8.3 and section 6.8.4). Based on these analyses, an object oriented building data model was established for the prototype (section 7.5).

An object oriented paradigm was adopted for the design and implementation of MultiCAD because of its advantages in key areas of software engineering (section 7.4). MultiCAD is designed as an MS-Windows application. The design and implementation of this prototype are aimed at meeting the requirements of an integrated, interactive and user friendly building design system (Chapter 7).

The MultiCAD prototype is implemented in the *ACTOR* programming environment. Testing and evaluation demonstrate that MultiCAD largely meets its specification set out at the beginning of the development process. In addition, it successfully illustrates the data exchange with the Integrated Data Model (IDM) of the COMBINE project (section 7.8).

MultiCAD has been compiled into a software package with full installation routine support (figure 9.1). It has been demonstrated in two international seminars in Germany and in the UK and is regarded as a step forward in the provision of design support for architects in practice (Lockley, 1993). The Building Research Establishment in the UK has expressed an interest in promoting MultiCAD as a commercial software product for distribution to architects.



Figure 9.1 The completed MultiCAD prototype

Technology transfer in the building industry is a very broad subject, realistically individual research project can only investigate a subset of it. This study identified and focused on one important link of this transfer process, architectural design support. Based on an analysis of others' studies on architectural design and design support, the objective of this study was further specified as exploring the potential of computer based Integrated Building Design System (IBDS).

It is not feasible to develop an IBDS which is suitable for architects to use in practice in the short term, given the huge task involved. Instead several issues were identified as essential for an IBDS, i.e., how to support multiple design tasks, how to maintain information exchange between different tasks, how to make the IBDS responsive to designers' requests, etc. MultiCAD was developed to explore these issues. While it is successful in demonstrating the principle of an IBDS, its ultimate effectiveness in architectural design support and technology transfer as a whole has to be tested by architects in practice. Before that can happen, several areas need to be addressed.

9.2 Issues for Future Study

The discussion in Chapter 3 revealed that there is no agreement on the design process model in the research community. Further study is needed in this area in the longer term, when the next generation of design support system, Intelligent Integrated Building Design System (IIBDS), is contemplated. An IIBDS differs from an IBDS in that it has intelligence support of design knowledge, in other words, when the user makes a decision or raises a request, the IIBDS is able to make consequent decisions in the design context. For example, if a designer sets a daylight requirement level for a room, an appropriate IIBDS would be capable of determining whether to change the opening area, window specification, internal surface material or a combination of changes. In an IBDS, these decisions are in the hand of the user. In order to realise the goal of the IIBDS, a further understanding of the design process and design knowledge are needed, which are unlikely to be achieved in the foreseeable future. In the meantime, there are scopes for the development of IBDS based on the current understanding of architectural design, furthermore an IBDS is a necessary step towards IIBDS.

For the development of an IBDS, five areas can be identified for further study based on the MultiCAD experience, they are data modelling, graphic interface, persistent data storage, system management and integration with external design tools.

- **Data modelling:** MultiCAD demonstrated that a data model is essential for an IBDS. The data exchange can only be effectively realised through a common data model which is able to support data views of all the design tools in the system. This data model is a neutral description of the building and its surrounding environment, it needs to be independent from any individual design tool in order to support a wide range of applications. However, it has been acknowledged that a complete building data model covering every aspect is not achievable in the short term. Instead, data modelling has to adopt an incremental approach. The MultiCAD's data model addressed construction and spatial system of the building. This data model needs to be consolidated and expanded to cover more aspects of the building, i.e., lighting, structural, costing, etc. The enhancement of the data model will provide the basis for supporting more design tools in the system and for integrating with other tools outside the system.
- **Graphic interface:** Geometry and topology are very important aspects of the building design. Architects have to make decisions on building layout and coordinates for all the building elements. Typical tasks include specifying room size and position, changing room size, deciding window opening size and position, etc. MultiCAD demonstrated how the user performs the building layout design using the simple functions offered by its CAD tool, and the information is stored in the data model. However, the functions are too limited and a more robust industry standard CAD tool is needed in order to enable architects to use an IBDS in practice. The main problem in this area is that most of the major CAD systems, i.e., AutoCAD and Intergraph Microstation, do not have full geometry and topology data model support. Consequently, when data from a richer building data model is loaded into the CAD system, some information will be lost. One possible solution to this problem is to retain all the data in the building data model and to use the CAD system simply as a tool of displaying and editing the geometry and topology aspect of the model. Adopting this strategy, efforts are needed on the mapping of entities in the CAD system with entities in the building data model, and the CAD functions with the data model entity's behaviours.
- **Persistent data storage:** Building design is usually a lengthy process. The information provided by designers should be stored so that it can be retrieved at a later stage. Therefore, an IBDS must provide persistent storage facilities to record the information when its operation terminates. A database system especially an

Object Oriented Database (OODB) system is needed for this task. The advantages of using an OODB also include services, such as information distribution through computer network, disciplined information sharing, recovery from unexpected system failures, etc. A number of issues have to be addressed in this area. For example, when an object is stored, it is created in a database. When it is no longer needed, it has to be removed from the database. Otherwise, the database will keep growing in size until it fills up the computer memory. This raises additional requirements on the data modelling, in which relationships between objects have to be specified for the purpose of object creation and deletion. For example, when a building object is removed from a database, all the room objects associated with the building should also be removed. On the other hand, when a room object is removed, the building object does not necessarily have to be removed from the database.

- **System management:** Several system management issues have been highlighted in the MultiCAD prototype which are not fully resolved, including concurrency control, object versioning, design history tracing, as well as the provision of default data.

One objective of an IBDS is the data sharing between different tools in the system. When a data item is shared by two tools, any change made by one tool should be notified to the other tool. Because both of the tools can make changes on the same data, concurrency control is necessary to prevent conflicting requests which can cause data inconsistency. Object oriented database systems usually solve this problem using a transaction lock control. At any given moment, only one tool is allowed to make changes on a shared data item, other tools have to wait for their turn.

In some data sharing situations when some tools need the data for lengthy evaluations, concurrent access might not be an ideal solution, object versioning is a more appropriate alternative. In object versioning, instead of making changes directly on the original data, each tool takes a version or copy of the data and makes changes on the copy. An issue needs to be addressed is that how to control the merging of all the different versions into one object.

Design history tracing is to provide the user with the possibility of going backward in the system operation sequence to review changes being made and to undo some changes. To achieve this, an IBDS must provide an efficient way of keeping history records of the system operation.

MultiCAD provides the user with default data when a building is instantiated. While it is a successful strategy in terms of system operation efficiency, there is a potential danger for confusion if the user is unaware of any undesired default data. Therefore, a clear control mechanism is needed to remind the user about those default data.

- **Integration with external design tools:** No single IBDS could possibly provide all the design support tools needed by architects. Many tools from different sources will continually be used in the architectural design practice. Therefore, there is a need for the wide integration between these tools in order to exchange design information. The development of ISO STEP/EXPRESS standard offers guidance and generic resources in this respect.

The participation in the EEC funded COMBINE 2 project provides an opportunity for the further study of these issues.

APPENDIX A

I/O DATA DOCUMENTATION FOR DAYLIGHT CALCULATION

Formatted I/O data documentation for Average daylight factor calculation

START_FORM:AVERAGE DAYLIGHT

1- Program or Calculation Method

1.1-Name or Identifier

Average daylight factor

1.2-version

version 1

1.3-Date of release

1987

1.4-Contact Name and address

CIBSE

Delta House

222 Balham High Road

London SW12 9BS

1.5-Brief Description

This is calculation for average daylight level in a room.

1.6-Main objects

site, room, window, internal surface.

1.7-Further information

no

2- Purpose

2.1- Definition

This is a calculation of the average daylight factor for a room with vertical window(s).

2.2- Rules for interpretation of results (if any)

Compare against CIBSE standard requirements

3- ID code for program-purpose

3.1- IDCODE

AVERAGE-DF

START_DATA:AVERAGE-DF_I_01

4- Object (related to input data)

4.1-Name or Identifier

site

4.2-Brief description (if needed)

-

4.3-Further Information

no

5- Input Data

5.1-Name or Identifier

location type.

5.2-Description

type of the location of the site, e.g. Clean, Industrial, Very dirty.

5.3-How used (purpose of use)

used to decide correction factor for the dirty on glazing.

5.4-How Derived (method)

associated with site

5.5-Related Objects (if any) or specific links to other data

site
 5.6-Type (format- real, integer, string, graph, etc.)
 string
 5.7-Units
 no
 5.8-Range
 -
 5.9-Default value (if any)
 no
 5.10- Source (of value, range etc.)
 Window Design of CIBSE. The value is one of the following three: Clean,
 Industrial, Very dirty.
 5.11- Further Information
 no
 START_DATA:AVERAGE-DF_I_02
 4- Object (related to input data)
 4.1-Name or Identifier
 internal surface
 4.2-Brief description (if needed)
 internal surface including that of walls, floor, ceiling.
 4.3-Further Information
 no
 5- Input Data
 5.1-Name or Identifier
 A
 5.2-Description
 surface area.
 5.3-How used (purpose of use)
 used to decide the total internal area of the room and the area weighted average light
 reflectance.
 5.4-How Derived (method)
 associated with room geometry
 5.5-Related Objects (if any) or specific links to other data
 no
 5.6-Type (format- real, integer, string, graph, etc.)
 real
 5.7-Units
 square meters
 5.8-Range
 greater than 0.0
 5.9-Default value (if any)
 no
 5.10- Source (of value, range etc.)
 no
 5.11- Further Information
 no
 START_DATA:AVERAGE-DF_I_03
 4- Object (related to input data)
 4.1-Name or Identifier
 internal surface
 4.2-Brief description (if needed)
 internal surface including that of walls, floor, ceiling, window glazing and window
 frame.

4.3-Further Information

no

5- Input Data

5.1-Name or Identifier

R

5.2-Description

diffuse light reflectance.

5.3-How used (purpose of use)

used to decide the area weighted average light reflectance.

5.4-How Derived (method)

associated with material and condition of the internal surface.

5.5-Related Objects (if any) or specific links to other data

no

5.6-Type (format- real, integer, string, graph, etc.)

real

5.7-Units

no

5.8-Range

0.0 to 1.0

5.9-Default value (if any)

no

5.10- Source (of value, range etc.)

publications on material properties.

5.11- Further Information

no

START_DATA:AVERAGE-DF_I_04

4- Object (related to input data)

4.1-Name or Identifier

window

4.2-Brief description (if needed)

-

4.3-Further Information

no

5- Input Data

5.1-Name or Identifier

glazing area

5.2-Description

net glazing area in the window opening excluding window frame area.

5.3-How used (purpose of use)

used to calculate average daylight factor.

5.4-How Derived (method)

associated with window design.

5.5-Related Objects (if any) or specific links to other data

no

5.6-Type (format- real, integer, string, graph, etc.)

real

5.7-Units

square meters

5.8-Range

greater than 0.0

5.9-Default value (if any)

no

5.10- Source (of value, range etc.)

no

5.11- Further Information

no

START_DATA:AVERAGE-DF_I_05

4- Object (related to input data)

4.1-Name or Identifier

window

4.2-Brief description (if needed)

-

4.3-Further Information

no

5- Input Data

5.1-Name or Identifier

frame area

5.2-Description

net frame area in the window opening.

5.3-How used (purpose of use)

used to calculate average daylight factor.

5.4-How Derived (method)

associated with window design.

5.5-Related Objects (if any) or specific links to other data

no

5.6-Type (format- real, integer, string, graph, etc.)

real

5.7-Units

square meters

5.8-Range

greater than 0.0

5.9-Default value (if any)

no

5.10- Source (of value, range etc.)

no

5.11- Further Information

no

START_DATA:AVERAGE-DF_I_06

4- Object (related to input data)

4.1-Name or Identifier

window

4.2-Brief description (if needed)

-

4.3-Further Information

no

5- Input Data

5.1-Name or Identifier

T

5.2-Description

diffuse transmittance of glazing material

5.3-How used (purpose of use)

used to calculate average daylight factor.

5.4-How Derived (method)

from related publications.

5.5-Related Objects (if any) or specific links to other data

no

5.6-Type (format- real, integer, string, graph, etc.)
 real

5.7-Units
 no

5.8-Range
 0.0 to 1.0

5.9-Default value (if any)
 no

5.10- Source (of value, range etc.)
 Window Design of CIBSE or other publication on glazing material properties.

5.11- Further Information
 no

START_DATA:AVERAGE-DF_I_04

4- Object (related to input data)

4.1-Name or Identifier
 window

4.2-Brief description (if needed)
 -

4.3-Further Information
 no

5- Input Data

5.1-Name or Identifier
 θ

5.2-Description
 visible sky angle

5.3-How used (purpose of use)
 used to calculate average daylight factor.

5.4-How Derived (method)
 associated with window design.

5.5-Related Objects (if any) or specific links to other data
 no

5.6-Type (format- real, integer, string, graph, etc.)
 real

5.7-Units
 degree

5.8-Range
 0.0 to 90.0

5.9-Default value (if any)
 no

5.10- Source (of value, range etc.)
 window geometry design

5.11- Further Information
 for the method of measurement of the visible sky angle see Window Design of CIBSE.

START_DATA:AVERAGE-DF_O_01

6- Object (related to output data)

6.1- Name or Identifier
 room

6.2- Brief description (if needed)
 -

6.3- Further Information
 no

7- Output Data

7.1- Name or Identifier
DF

7.2- Description
average daylight factor. It is the ratio of internal average illuminance (a spatial average over the working plane) to external global horizontal illuminance under the assumption of CIE overcast sky condition.

7.3- Related Objects (if any) or specific links to other data
-

7.4- Type (format- real, integer, string, graph, etc.)
real

7.5- Units
no

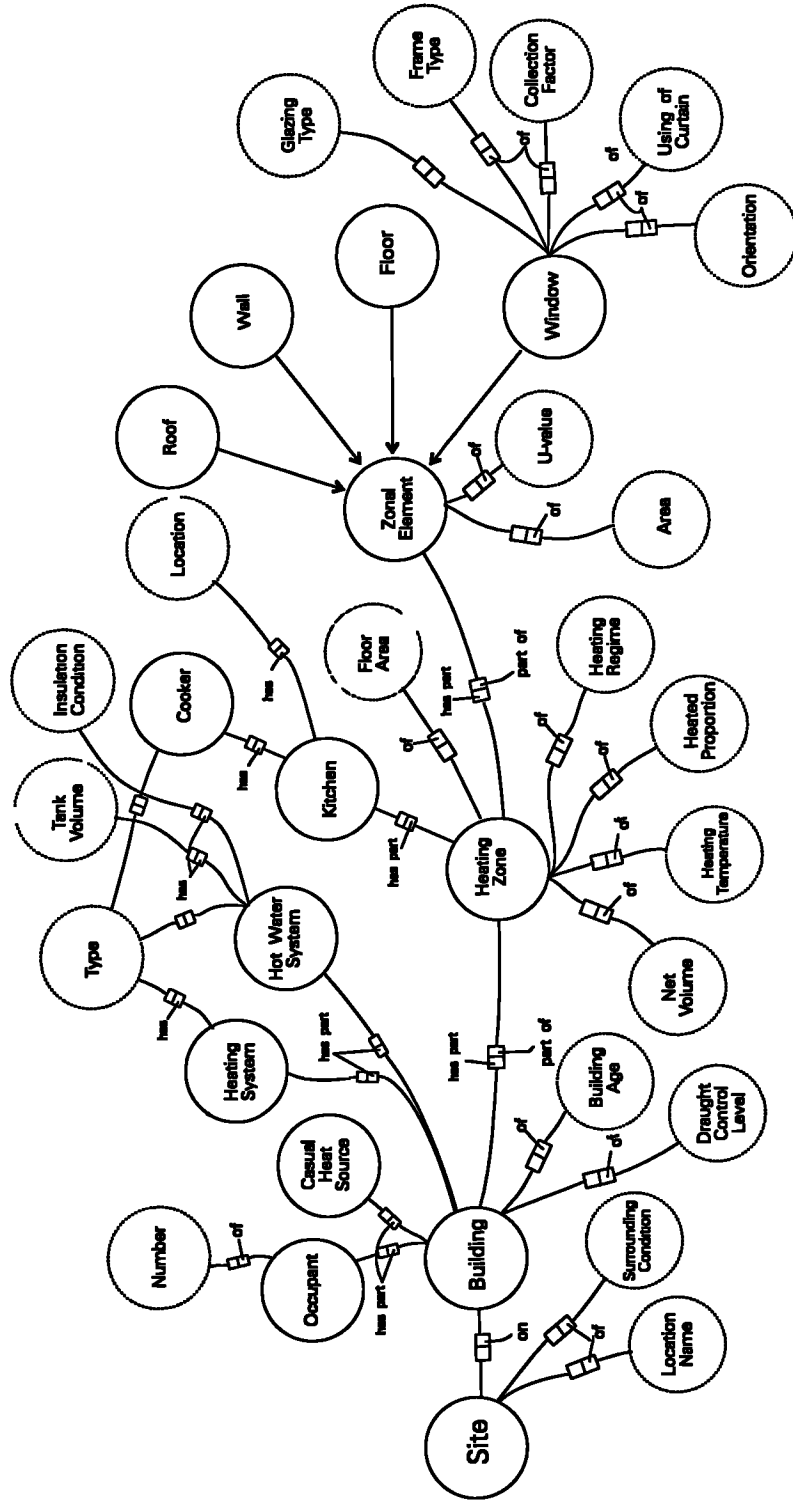
7.6- Range
greater than 0.0

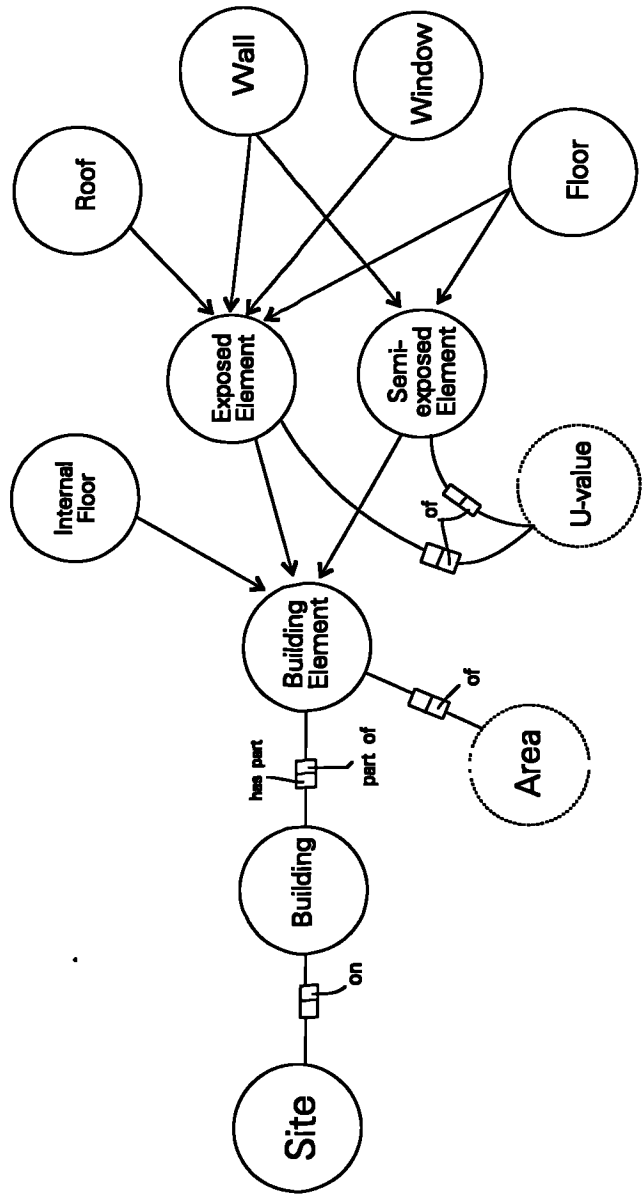
7.7- Further Information
no

END_FORM:AVERAGE-DF

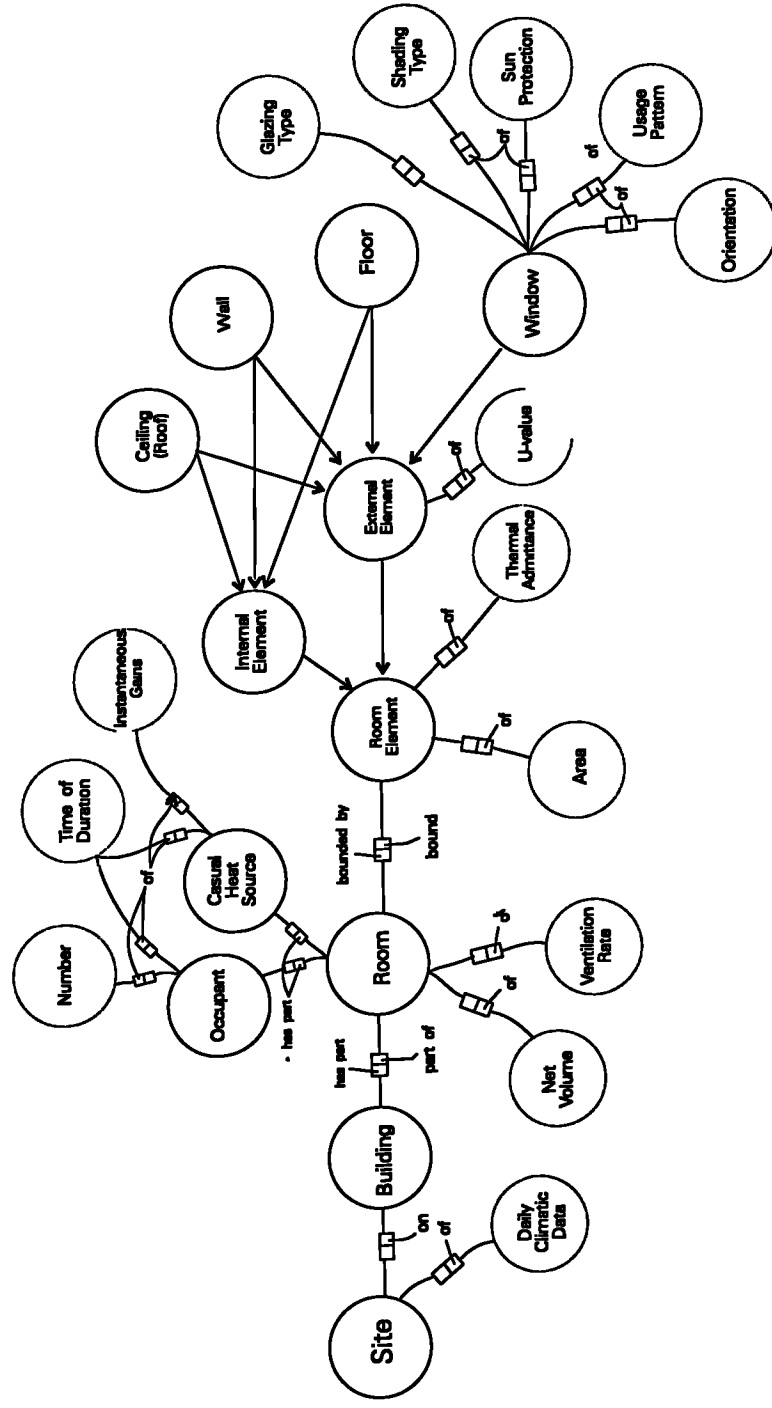
APPENDIX B

NIAM ANALYSIS

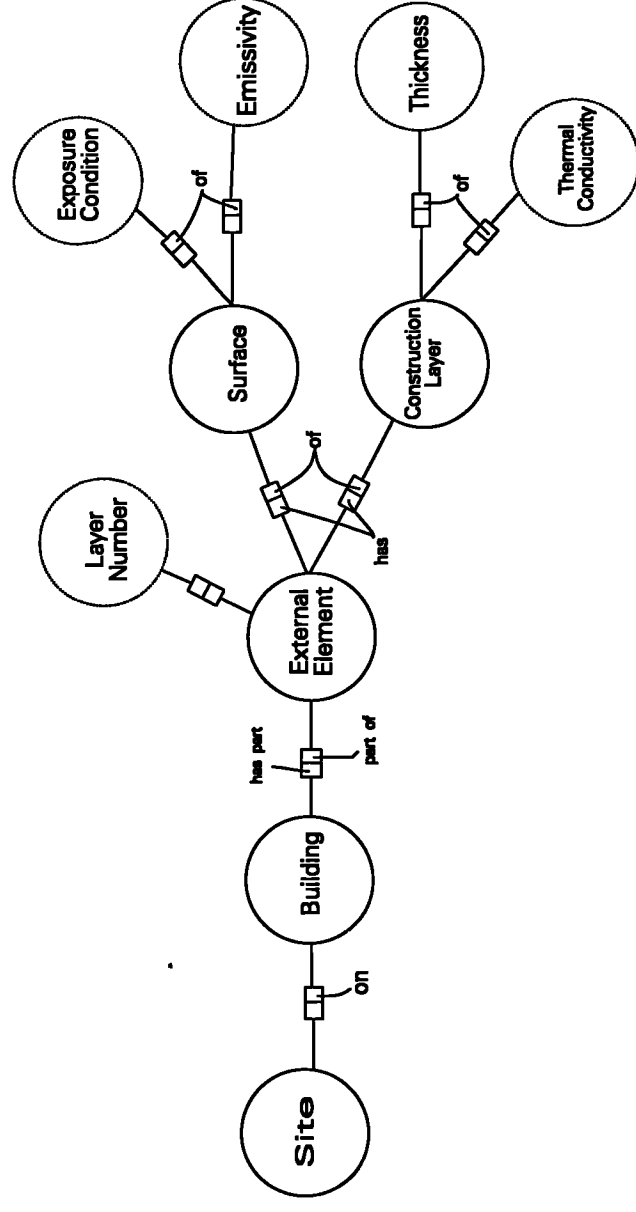




NIAM diagram of building model for compliance checking



NIAM diagram of building model for overheating



NIAM diagram of building model for elemental U-value

APPENDIX C

EXPRESS SCHEMA OF THE IDM

EXPRESS Schema of the IDM

SCHEMA IDM;

ENTITY geometry

SUPERTYPE OF (ONEOF (axis2_placement,
cartesian_coordinate_system, face, point, position, shell, surface, tangible_object_geometry,
vector));

id_name : STRING ;

is_geometry_of : building_object ;

UNIQUE

id_name ;

END_ENTITY ;

ENTITY cartesian_point

SUBTYPE OF (point) ;

x_coordinate : REAL ;

y_coordinate : REAL ;

z_coordinate : REAL ;

END_ENTITY ;

ENTITY point

SUPERTYPE OF (cartesian_point)

SUBTYPE OF (geometry) ;

END_ENTITY ;

ENTITY axis2_placement

SUBTYPE OF (geometry) ;

has_cartesian_point : cartesian_point ;

has_y_reference_direction : direction ;

has_z_axis_direction : direction ;

END_ENTITY ;

ENTITY vector

SUPERTYPE OF (direction)

SUBTYPE OF (geometry) ;

END_ENTITY ;

ENTITY direction

SUPERTYPE OF (ONEOF (horizontal_x_axis, horizontal_y_axis,
vertical_z_axis))

SUBTYPE OF (vector) ;

x : REAL ;

y : REAL ;

z : REAL ;

END_ENTITY ;

ENTITY horizontal_x_axis

SUBTYPE OF (direction) ;

END_ENTITY ;

```
ENTITY horizontal_y_axis
  SUBTYPE OF (direction) ;
END_ENTITY ;
```

```
ENTITY vertical_z_axis
  SUBTYPE OF (direction) ;
END_ENTITY ;
```

```
ENTITY cartesian_coordinate_system
  SUPERTYPE OF ( building_coordinate_system)
  SUBTYPE OF (geometry) ;
END_ENTITY ;
```

```
ENTITY building_coordinate_system
  SUBTYPE OF (cartesian_coordinate_system) ;
  y_axis_azimuth : REAL ;
  has_horizontal_x_axis : direction ;
  has_horizontal_y_axis : direction ;
  has_vertical_z_axis : direction ;
  has_point : point ;
  WHERE
    ( 0. <= y_axis_azimuth ) AND ( y_axis_azimuth <= 360. ) ;
END_ENTITY ;
```

```
ENTITY box_domain
  SUPERTYPE OF ( building_box_domain) ;
  id_name : STRING ;
  z_min : REAL ;
  y_min : REAL ;
  x_min : REAL ;
  z_max : REAL ;
  y_max : REAL ;
  x_max : REAL ;
  UNIQUE
    id_name ;
END_ENTITY ;
```

```
ENTITY building_box_domain
  SUBTYPE OF (box_domain) ;
  x_length : REAL ;
  y_length : REAL ;
  z_length : REAL ;
END_ENTITY ;
```

```
(*
ENTITY void
  SUBTYPE OF (solid_model) ;
END_ENTITY ;
```

```
ENTITY volume_shape
  SUBTYPE OF (shape) ;
```

```

    kind : STRING ;
END_ENTITY ;
*)

```

```

ENTITY element_shell
  SUBTYPE OF (shell) ;
  has_side2 : face ;
  has_side1 : face ;
  UNIQUE
    has_side2 ;
    has_side1 ;
END_ENTITY ;

```

```

ENTITY shell
  SUPERTYPE OF ( ONEOF ( element_shell, space_shell ) )
  SUBTYPE OF (geometry) ;
  is_made_of : SET OF face ;
END_ENTITY ;

```

```

ENTITY face
  SUPERTYPE OF ( ONEOF ( subface, space_face ) )
  SUBTYPE OF (geometry) ;
  area : REAL ;
  azimuth : REAL ;
  tilt : REAL ;
  has_list_of : SET OF point ;
  WHERE
    ( 0. <= azimuth ) AND ( azimuth <= 360. ) ;
    ( 0. <= tilt ) AND ( tilt <= 180. ) ;
END_ENTITY ;

```

```

ENTITY space_face
  SUBTYPE OF (face) ;
  is_attached_to : SET OF subface ;
END_ENTITY ;

```

```

ENTITY subface
  SUBTYPE OF (face) ;
  global_area : REAL ;
  net_area : REAL ;
  perimeter : REAL ;
  x_length : REAL ;
  y_length : REAL ;
  is_part_of : side ;
  faces_space : space ;
  is_attached_to : space_face ;
  WHERE
    net_area > 0. ;
    perimeter > 0. ;
    x_length > 0 ;
    y_length > 0 ;
END_ENTITY ;

```

```

ENTITY surface
  SUPERTYPE OF ( element_surface)
  SUBTYPE OF (geometry) ;
END_ENTITY ;

```

```

ENTITY element_surface
  SUBTYPE OF (surface) ;
  area : REAL ;
  tilt : REAL ;
  x_length : REAL ;
  y_length : REAL ;
  has_position : axis2_placement ;
  WHERE
    area > 0 ;
    ( 0. <= tilt ) AND ( tilt <= 180. ) ;
    x_length > 0 ;
    y_length > 0 ;
END_ENTITY ;

```

```

ENTITY element_geometry
  SUBTYPE OF (tangible_object_geometry) ;
  global_area : REAL ;
  has_a_2D_aspect : element_surface ;
  has_a_shell : shell ;
END_ENTITY ;

```

```

ENTITY building_object
  SUPERTYPE OF ( ONEOF ( enclosure_element, hole, space) ) ;
  id_name : STRING ;
  has_type : STRING ;
  UNIQUE
    id_name ;
END_ENTITY ;

```

```

ENTITY enclosure_element
  SUPERTYPE OF ( ONEOF ( facade, floor,
  outside_wall, roof, wall,
  opening) )
  SUBTYPE OF (building_object) ;
  is_part_of : OPTIONAL building_fabric_system ;
  is_made_of : OPTIONAL element_construction ;
  has_for_geometry : element_geometry ;
  contains : SET OF hole ;
  has_side_2 : side ;
  delimits : OPTIONAL SET OF space ;
END_ENTITY ;

```

```

ENTITY facade
  SUBTYPE OF (enclosure_element) ;
  area : OPTIONAL REAL ;
  azimuth : OPTIONAL REAL ;
  is_composed_of : OPTIONAL SET OF outside_wall ;
  WHERE

```

(0. <= azimuth) AND (azimuth <= 360.) ;
END_ENTITY ;

ENTITY floor
SUBTYPE OF (enclosure_element) ;
END_ENTITY ;

ENTITY outside_wall
SUBTYPE OF (enclosure_element) ;
area : OPTIONAL REAL ;
slope : OPTIONAL REAL ;
END_ENTITY ;

ENTITY roof
SUBTYPE OF (enclosure_element) ;
area : OPTIONAL REAL ;
slope : OPTIONAL SET OF REAL ;
is_composed_of : OPTIONAL SET OF enclosure_element ;
END_ENTITY ;

ENTITY wall
SUPERTYPE OF (ONEOF (internal_wall))
SUBTYPE OF (enclosure_element) ;
END_ENTITY ;

ENTITY internal_wall
SUBTYPE OF (wall) ;
END_ENTITY ;

ENTITY opening
SUPERTYPE OF (ONEOF (door, window))
SUBTYPE OF (enclosure_element) ;
area : REAL ;
height : REAL ;
width : REAL ;
is_made_of_frame : frame ;
is_made_of_glazing : glazing ;
is_located_in : hole ;
has_relative_position : position_in_hole ;
END_ENTITY ;

ENTITY window
SUBTYPE OF (opening) ;
END_ENTITY ;

ENTITY door
SUBTYPE OF (opening) ;
END_ENTITY ;

ENTITY hole
SUBTYPE OF (building_object) ;
has_geometry : element_geometry ;
has_hole_position : hole_position ;

```

contains_enclosure_element : SET OF enclosure_element ;
contains_opening : SET OF opening ;
is_located_in : enclosure_element ;
END_ENTITY ;

```

```

ENTITY position
  SUPERTYPE OF ( ONEOF ( position_in_hole, hole_position) )
  SUBTYPE OF (geometry) ;
END_ENTITY ;

```

```

ENTITY position_in_hole
  SUBTYPE OF (position) ;
  position_hz_in_the_element : REAL ;
  position_vert_in_the_element : REAL ;
  relative_position_in_hole : STRING ;
  setback : REAL ;
  sill_height : REAL ;
  of_opening : opening ;
  is_related_to : hole ;
  oriented : side ;
  opening_direction : SET OF side ;
END_ENTITY ;

```

```

ENTITY hole_position
  SUBTYPE OF (position) ;
  shape_type : STRING ;
  has_axis2_placement : axis2_placement ;
  is_related_to : enclosure_element ;
END_ENTITY ;

```

```

(*)
ENTITY shape
  SUPERTYPE OF ( ONEOF ( surface_shape, volume_shape) )
  SUBTYPE OF (geometry) ;
  parameter_number : INTEGER ;
  shape_type : STRING ;
  WHERE
    parameter_number > 1 ;
END_ENTITY ;

```

```

ENTITY rectangle
  SUBTYPE OF (surface_shape) ;
  x_length : REAL ;
  y_length : REAL ;
  WHERE
    x_length > 0 ;
    y_length > 0 ;
END_ENTITY ;

```

```

*)

```

```

ENTITY side;
  id_name : STRING ;

```

```

azimuth : REAL ;
has_geometry : face ;
contains : SET OF subface ;
is_side_of : enclosure_element ;
UNIQUE
id_name ;
is_side_of ;
WHERE
( 0. <= azimuth ) AND ( azimuth <= 360. ) ;
END_ENTITY ;

```

```

(*)
ENTITY size;
id_name : STRING ;
size_parameter : SET OF REAL ;
UNIQUE
id_name ;
WHERE
size_parameter > 0. ;
END_ENTITY ;

```

```

ENTITY solid_model
SUPERTYPE OF ( void)
SUBTYPE OF (geometry) ;
END_ENTITY ;
*)

```

```

ENTITY space_geometry
SUPERTYPE OF ( building_geometry)
SUBTYPE OF (tangible_object_geometry) ;
floor_level : REAL ;
height : REAL ;
length : REAL ;
width : REAL ;
has_position_in : building_coordinate_system ;
has_face : face ;
is_delimited_by : space_shell ;
END_ENTITY ;

```

```

ENTITY space_shell
SUBTYPE OF (shell) ;
is_limits_of : space_geometry ;
END_ENTITY ;

```

```

(*)
ENTITY surface_shape
SUPERTYPE OF ( rectangle)
SUBTYPE OF (shape) ;
planear_form : STRING ;
has_type : STRING ;
END_ENTITY ;
*)

```


ENTITY tangible_object_geometry
SUPERTYPE OF (ONEOF (element_geometry, space_geometry))
SUBTYPE OF (geometry) ;
has_axis2_placement : axis2_placement ;
END_ENTITY ;

ENTITY space
SUPERTYPE OF (ONEOF (circulation_space, room, technical_space, zone))
SUBTYPE OF (building_object) ;
floor_area : OPTIONAL REAL ;
gross_area : OPTIONAL REAL ;
height : OPTIONAL REAL ;
volume : OPTIONAL REAL ;
walls_area : OPTIONAL REAL ;
is_part_of_building_spatial_system : OPTIONAL building_spatial_system ;
is_delimited_by : OPTIONAL SET OF enclosure_element ;
is_adjacent_to_ground : OPTIONAL ground ;
has_opening : OPTIONAL SET OF opening ;
is_adjacent_to_outside : OPTIONAL outside ;
is_adjacent_to_space : OPTIONAL SET OF space ;
has_space_geometry : OPTIONAL space_geometry ;
is_part_of_zone : OPTIONAL zone ;
includes_zone : OPTIONAL SET OF zone ;
WHERE
floor_area > 0 ;
gross_area > 0 ;
volume > 0. ;
END_ENTITY ;

ENTITY circulation_space
SUPERTYPE OF (ONEOF (horizontal_circulation_space))
SUBTYPE OF (space) ;
END_ENTITY ;

ENTITY horizontal_circulation_space
SUPERTYPE OF (corridor)
SUBTYPE OF (circulation_space) ;
END_ENTITY ;

ENTITY corridor
SUBTYPE OF (horizontal_circulation_space) ;
END_ENTITY ;

ENTITY technical_space
SUPERTYPE OF (hvac_technical_space)
SUBTYPE OF (space) ;
END_ENTITY ;

ENTITY hvac_technical_space
SUBTYPE OF (technical_space) ;
END_ENTITY ;

ENTITY room

SUPERTYPE OF (ONEOF (office))
SUBTYPE OF (space) ;
END_ENTITY ;

ENTITY office
SUBTYPE OF (room) ;
END_ENTITY ;

ENTITY zone
SUBTYPE OF (space) ;
is_composed_of : OPTIONAL SET OF space ;
is_subpart_of : OPTIONAL space ;
END_ENTITY ;

ENTITY outside;
id_name : STRING ;
is_adjacent_to : OPTIONAL SET OF space ;
UNIQUE
id_name ;
END_ENTITY ;

ENTITY ground;
id_name : STRING ;
is_adjacent_to : OPTIONAL SET OF space ;
UNIQUE
id_name ;
END_ENTITY ;

ENTITY measure
SUPERTYPE OF (ONEOF (characteristic));
id_name : STRING ;
value : NUMBER ;
UNIQUE
id_name ;
END_ENTITY ;

ENTITY characteristic
SUPERTYPE OF (ONEOF (physical_characteristic,thermal_characteristic))
SUBTYPE OF (measure) ;
kind : OPTIONAL STRING ;
name : OPTIONAL STRING ;
has_type : OPTIONAL STRING ;
END_ENTITY ;

ENTITY thermal_characteristic
SUPERTYPE OF (ONEOF (emissivity, reflectance,specific_heat, thermal_transmittance,
conductivity))
SUBTYPE OF (characteristic) ;
END_ENTITY ;

ENTITY physical_characteristic

SUPERTYPE OF (mass_density)
SUBTYPE OF (characteristic) ;
END_ENTITY ;

ENTITY conductivity
SUBTYPE OF (thermal_characteristic) ;
END_ENTITY ;

ENTITY specific_heat
SUBTYPE OF (thermal_characteristic) ;
END_ENTITY ;

ENTITY mass_density
SUBTYPE OF (physical_characteristic) ;
END_ENTITY ;

ENTITY emissivity
SUBTYPE OF (thermal_characteristic) ;
surface_state : OPTIONAL SET OF STRING ;
END_ENTITY ;

ENTITY reflectance
SUBTYPE OF (thermal_characteristic) ;
surface_condition : OPTIONAL STRING ;
END_ENTITY ;

ENTITY thermal_transmittance
SUBTYPE OF (thermal_characteristic) ;
END_ENTITY ;

ENTITY material
SUPERTYPE OF (finish_material) ;
id_name : STRING ;
has_conductivity : OPTIONAL conductivity ;
has_emissivity : OPTIONAL emissivity ;
has_reflectance : OPTIONAL reflectance ;
has_specific_heat : OPTIONAL specific_heat ;
has_mass_density : OPTIONAL mass_density ;
has_thermal_transmittance : OPTIONAL thermal_transmittance ;
UNIQUE
id_name ;
END_ENTITY ;

ENTITY finish_material
SUBTYPE OF (material) ;
colour : STRING ;
END_ENTITY ;

ENTITY layer
SUPERTYPE OF (finish) ;

```

id_name : STRING ;
insulation_layer : OPTIONAL STRING ;
K_coeff : OPTIONAL REAL ;
layer_rank : OPTIONAL INTEGER ;
layer_thickness : OPTIONAL REAL ;
U_value : OPTIONAL REAL ;
is_made_of : OPTIONAL material ;
UNIQUE
id_name ;
END_ENTITY ;

```

```

ENTITY finish
SUBTYPE OF (layer) ;
emissivity : REAL ;
finish_thickness : REAL ;
has_reflectance : reflectance ;
WHERE
( 0. <= emissivity ) AND ( emissivity <= 1. ) ;
END_ENTITY ;

```

```

ENTITY construction_type
SUPERTYPE OF ( ONEOF ( frame_type,
glazing_type ) );
id_name : STRING ;
air_layer_rank : OPTIONAL INTEGER ;
air_layer_resistance : OPTIONAL REAL ;
insulation_layer_rank : OPTIONAL SET OF INTEGER ;
layer_number : OPTIONAL INTEGER ;
light_transmittance : OPTIONAL REAL ;
thermal_capacity_1 : OPTIONAL SET OF REAL ;
thermal_capacity_2 : OPTIONAL SET OF REAL ;
U_value : OPTIONAL REAL ;
U_value_btu : OPTIONAL REAL ;
user_definition : OPTIONAL STRING ;
is_referenced_to : OPTIONAL catalog ;
is_used_for : OPTIONAL SET OF element_construction ;
is_composed_of : OPTIONAL SET OF layer ;
UNIQUE
id_name ;
WHERE
air_layer_resistance > 0 ;
layer_number > 0 ;
( 0. <= light_transmittance ) AND ( light_transmittance <= 1. ) ;
thermal_capacity_1 > 1. ;
thermal_capacity_2 > 1. ;
( 0.0009999999 <= U_value_btu ) AND ( U_value_btu <= 20. ) ;
END_ENTITY ;

```

```

ENTITY catalog;
id_name : STRING ;
Catalog_name : STRING ;
UNIQUE

```

id_name ;
END_ENTITY ;

ENTITY frame_type
SUBTYPE OF (construction_type) ;
frame_form : OPTIONAL STRING ;
frame_material : OPTIONAL STRING ;
opening_type : OPTIONAL STRING ;
panel_number : OPTIONAL INTEGER ;
opening_direction : OPTIONAL STRING ;
WHERE
(1 <= panel_number) AND (panel_number <= 10) ;
END_ENTITY ;

ENTITY glazing_type
SUBTYPE OF (construction_type) ;
glass_number : OPTIONAL INTEGER ;
WHERE
(1 <= glass_number) AND (glass_number <= 4) ;
END_ENTITY ;

ENTITY tangible_object
SUPERTYPE OF (ONEOF (frame, glazing));
id_name : STRING ;
UNIQUE
id_name ;
END_ENTITY ;

ENTITY frame
SUBTYPE OF (tangible_object) ;
openable_area : REAL ;
has_frame_type : frame_type ;
END_ENTITY ;

ENTITY glazing
SUBTYPE OF (tangible_object) ;
area : REAL ;
height : REAL ;
width : REAL ;
is_made_of : glazing_type ;
END_ENTITY ;

ENTITY element_construction;
id_name : STRING ;
insulation_layer_place : OPTIONAL INTEGER ;
insulation_required : OPTIONAL STRING ;
side_reference : OPTIONAL STRING ;
U_value_required : OPTIONAL REAL ;
has_type : OPTIONAL construction_type ;
is_construction_of : OPTIONAL SET OF enclosure_element ;

```

UNIQUE
  id_name ;
WHERE
  ( 0 <= insulation_layer_place ) AND
  ( insulation_layer_place <= 3 ) ;
END_ENTITY ;

```

```

ENTITY building_spatial_system;
  id_name : STRING ;
  is_aspect_of : building ;
  is_composed_of_space : SET OF space ;
UNIQUE
  id_name ;
END_ENTITY ;

```

```

ENTITY building;
  id_name : STRING ;
  has_building_local_site : building_local_site ;
  has_aspect : building_spatial_system ;
  is_part_of : construction_project ;
UNIQUE
  id_name ;
END_ENTITY ;

```

```

ENTITY building_fabric_system;
  id_name : STRING ;
  is_fabric_of : building ;
  is_composed_of : SET OF enclosure_element ;
  has_part_facade : SET OF facade ;
  has_part_roof : SET OF roof ;
UNIQUE
  id_name ;
END_ENTITY ;

```

```

ENTITY building_geometry
SUBTYPE OF (space_geometry) ;
  ground_level : REAL ;
  volume : REAL ;
  has_building_box_domain : building_box_domain ;
  has_building_coordinate_system : building_coordinate_system ;
  of_building : building ;
WHERE
  volume > 0 ;
END_ENTITY ;

```

```

ENTITY construction_project;
  id_name : STRING ;
  is_composed_of : SET OF building ;
  is_located_in : site ;
UNIQUE
  id_name ;
END_ENTITY ;

```

```
ENTITY site
  SUPERTYPE OF ( building_local_site );
  id_name : STRING ;
  is_composed_of : SET OF building_local_site ;
  contains : construction_project ;
  is_bounded_by : surroundings ;
  UNIQUE
    id_name ;
END_ENTITY ;
```

```
ENTITY building_local_site
  SUBTYPE OF (site) ;
  of_building : building ;
  is_part_of : site ;
END_ENTITY ;
```

```
ENTITY surroundings;
  id_name : STRING ;
  has_ground : ground ;
  of_site : site ;
  UNIQUE
    id_name ;
END_ENTITY ;
```

```
END_SCHEMA ;
```

APPENDIX D

CLASS REFERENCE FOR MULTICAD DATA MODEL

Class reference for the data model

This appendix is a reference list for related classes in the data model. It is presented as a supplement to the data model discussed in chapter 7. It documents the methods or behaviours of the object classes. No interface class is included in this list.

The information presented here is produced directly from the implementation source code. It is in the same format as that of the ACTOR class reference. For each class the information includes:

- Description of the class.
- Name of the class source file.
- Ancestor classes.
- Descendant classes.
- Inherited and defined instance and class variables.
- Header and description of class and object methods.

Building

Source file: BUILDING.CLS
Inherits from: Object
Inherited by: (no descendants)

Building Object

Instance variables:

rooms	all the rooms
constructions	all the construction
finishes	all the finishes
elements	all the elements
roomLinkers	network
nodes	BGMnetwork
site	building site
yearBuilt	year building was built
buildingType	string, eg Domestic
occupants	For efficiency
heatingSystem	For efficiency
hotWaterSystem	For efficiency
ventilationSystem	For efficiency
bredeZone	For efficiency

Class variables: (none)

Class methods:

functions(self)
return names of all the functions
readInVariables(self,obj,reader);
mapping with the IDM

Object methods:

addConstruction(self,construction)
add a construction type
addElement(self,elem,polygon)
add a set of nodes to the node network ensuring closure
addFinishes(self,finish)
add a finish type
addRoom(self,rm)
add a room to the building.
age(self)
return age of building eg 30 years old
airChangeRate(self)
airChangeRate in air changes per hour. Bredem-8 A.4.1 pp6-7
allExternalElements(self)
return a collection of all the external elements
annualHeatingEnergyUse(self)
annualHeatingEnergyUse returns annual energy used to heat building, in GJ
annualHotWaterEnergyUse(self)
annualHotWaterEnergyUse returns annual energy used for hot water, in GJ

bredeZone(self,zoneNum)
returns a collection of rooms corresponding to the zone, (1 or 2). Doesn't yet handle recursive rooms

bredeZone1(self)
return brede zone 1

bredeZone2(self)
return brede zone 2

buildingType(self)
return building type

constructions(self)
return a collection of construction used in the building.

dailyHotWaterEnergyUse(self)
daily Hot Water Energy Use drawn off each day, in Watts. Brede-8 p22 section 1

doors(self)
return all the door openings in the building.

effectiveAch(self)
effectiveAch gives the modified infiltration rate. Brede-8 p7

elements(self)
return all elements in the building.

energyLoss(self)
calculate energy lose of the building.

fabricLoss(self)
calculate the total building fabric loss sum ('U' * area) maybe? who cares or knows, maybe ESP In Watts per degree centigrade

finishes(self)
return all the element finishes in the building.

floorArea(self)
floorArea in square metres

gainLoadRatio(self, month)
gainLoadRatio Brede-8 A.4.4.3 p11

getInfo(self)
return info about Building

getPlanView(self)
return an ObjectGraphics Plan view

heatingSystem(self)
heatingSystem

hotWaterSystem(self)
hotWaterSystem

infiltrationRate(self)
Brede-8 air infiltration rate. Brede-8 pp 6-7. Depends upon the age of the dwelling, whether it is sheltered or exposed and whether the floor is suspended or solid

init(self)
Initialises the building objects

lightingGain(self)
lightingGain defined in Brede-8 A.4.2, p8

makeRoom(self,elems)
add a new room to the building. The argument is a collection of elements

meanExternalTemp(self,month)
meanExternalTemperature

monthlyFabricLoss(self,mon)
calculate monthly fabric heat loss.

monthlyHotWaterEnergyUse(self,month)
monthlyHotWaterEnergyUse defines the energy used to supply hot water in a given month, in GJ.
Bredem-8 A.5, pp22-23

monthlyVentLoss(self,mon)
calculate monthly ventilation heat loss.

monthlyWaterHeatingReqt(self, month)
waterHeatingReqt Bredem-8 A5.8 p23 Only valid if hot water is not instantaneous or produced from an immersion heater

name(self)
return name of the building

nearestPoint(self, point)
return point in the network nearest to the point sent. I don't think I can do this yet, there must some wonderful algorithm somewhere

nodes(self)
return nodes

notStorable(self)
Comment

occupants(self)
return number of occupants

region(self)
return the region for BREDEM calculation.

regionNumber(self)
return the region number for BREDEM calculation.

rooms(self)
return room collection

setSite(self,st)
set site to building

setupBuilding(self,rms)
set up a building using data mapped from STEP files.

setupOpenings(self,elem,ops)
set up the room1 and room2 of the openings.

setViewPlan(self)
return an ObjectGraphics Plan view

site(self)
return a building site

specificHeatLoss(self)
specificHeatLoss Bredem-8 A.4 p7

storeDefinitionOn(self, stream, table)
for data mapping with the IDM

thermalLoad(self,to,ti)
the building heat load

utilisationFactor(self, month)
utilisationFactor Bredem-8 A.4.4.3 p 12

volume(self)
 volume in cubic metres

windows(self)
 return all the window openings in the building.

yearBuilt(self)
 Year a building was built eg 1945

Room

Source file: ROOM.CLS

Inherits from: Object

Inherited by: BredemZone Site

General Room class

Class variables: (none)

Instance variables:

id	For data mapping purpose. SM
name	name of the room
lights	number
occupants	number
equipment	equipment in the room
roomLinkers	from the building
rooms	rooms in the space

Class methods:

functions(self)
 return the name of all the functions

readInVariables(self,obj,reader);
 for data mapping with the IDM

Object methods:

addEquipment(self,equip)
 add equipment to the room

addRoom(self,rm)
 add a room to the space.

areaAdmittance(self)
 total area weighted admittance of a room

averageDaylightFactor(self)
 return the average daylight factor according to CIBSE 1987 window design guide

avgWeightedReflectance(self)
 calculate area weighted average light reflectance

bpeMethods(self)
 Specifies useful BPE methods, with no parameters

bredemZone(self)
 room is in Bredem zone 1 or 2. Bredem-8 A.4 p6

connected(self)
 return all rooms which are connected to this one. method wrong and never used. SM

doors(self)
 return door objects of the room

editPicture(self,boundsRect)
return a plan picture

elements(self)
return a collection of all elements forming a room including those of rooms within rooms therefore this can be used to get all elements in a zone

equipment(self)
equipment

fabricLoss(self)
get all the elements bounding this room including sub-rooms and calculates the total fabric loss

finishes(self)
return finish of all elements of the room

firstExternalElement(self)
find an external element of a room

firstExternalWall(self)
find an external wall of a room.

floor(self)
return floor object of the room

floorArea(self)
temporary floorArea method. Checks for an element with z values all 0

getInfo(self)
getInfo

getPlanView(self)
return an ObjectGraphics Plan view

getSite(self)
get access to the site.

glazingLoss(self)
return heat loss from glazing in a room

hourlyAirToAirHeatGainSwing(self,hour)
modified for inf and multiplying by temperature

hourlyCasualGainSwing(self,hour)
hourly casual gain swing

hourlyEnvironmentalTemperature(self,hour)
hourly environmental temperature

hourlyInternalTemp(self)
calculate the hourly internal temperature for a room for 24 hours in a day.

hourlySolAirTempSwing(self,hour)
Comment

hourlySolarGainSwing(self,hour)
hourly solar gain swing

hourlyStructuralGainSwing(self,hour)
hourly structural gain swing

hourlySwingInEnvironmentalTemperature(self,hour)
hourly swing in environmental temperature

hourlyTotalSwingInHeatGain(self,hour)
hourly total swing in heat gain

id(self)
return identifier

information(self)
return brief information about the room

init(self)
initialise room object

initCasualSchedule(self)
for BREADMIT

isExternal(self)
return self if room is descendent of or is a site

isInternal(self)
return true if room is not external

isZone(self)
return whether a room is a single room or a zone

lights(self)
return a number

meanCasualGain(self)
mean casual gain

meanEnvironmentalTemperature(self)
mean Environmental Temperature

meanGain(self)
mean heat gain

meanInternalTemperature(self)
mean internal temperature

meanSolAirTemp(self)
mean solair temperature

meanSolarGain(self)
mean solar gain

name(self)
return name of the room

occupants(self)
return occupant number of the room

polyMesh(self)
return a polymesh that bounds the rooms volume

responseFactor(self)
response factor of the room

responseFactorAch(self,ach)
response factor of the room depending on air change rate

rooms(self)
return all the rooms if any

setName(self,id)
give a room an identity

setupRoom(self,elems)
addsa new room to the building. The argument is a collection of elements

setViewPlan(self)
modified plan view which stores graphic links

solarGain(self,building,month)
solarGain Bredem-8 A.4.3 p9-10 in kWh/day

specificHeatLoss(self, ach)
specific heat loss

storeDefinitionOn(self, stream, table)
for data mapping

surfaceArea(self)
surfaceArea in square metres

thermalLoad(self,ach,to,ti)
thermal load of the room

totalConductance(self)
total conductance

totalInternalAdmittance(self)
total Internal Admittance

totalSwingInHeatGain(self,hour)
total Swing In Heat Gain

ventilationLoss(self,ach)
ventilationLoss Bredem-8 p7

volume(self)
calculate volume of the room

wallArea(self)
return the total wall area for STEP data mapping.

walls(self)
return a collection of all elements forming a room including those of rooms within rooms therefore this can be used to get all elements in a zone

windows(self)
return all the window openings of the room

BredemZone

Source file: BREDEMZO.CLS

Inherits from: Object Room

Inherited by: (no descendants)

Class to handle Bredem Zones. Basically an OrderedCollection of rooms, created on the fly by extracting the correct rooms from the building, in method Building:BredemZone. There is no real need for it to be an OrderedCollection, could just be a Collection . Should descent from Room.
SM

Class variables: (none)

Instance variables:

id	(From class Room)
name	(From class Room)
lights	(From class Room)
occupants	(From class Room)
equipment	(From class Room)
roomLinkers	(From class Room)
rooms	(From class Room)
zoneNum	stores the zone number 1 or 2
building	stores the building object pointer

Class methods: (none)

Object methods:

backgroundHeatGain(self,month)
backgroundHeatGain Bredem-8 A.4.4.5 p 12

fabricLoss(self)
fabricLoss

floorArea(self)
floorArea in square metres

internalHeatGain(self)
internalHeatGain defined in Bredem-8 A.4.2 p9

lightingGain(self)
lightingGain Bredem-8 p8

meanInternalTemp(self,month)
meanInternalTemp Bredem-8 A.4.4.6 p12-13

monthlyFabricLoss(self,mon)
monthlyFabricLoss

monthlyVentLoss(self,mon)
monthlyVentLoss

netUsefulGain(self,month)
netUsefulGain Bredem-8 A.4.4.4 p 12

otherZone(self)
Return the otherZone. zoneNum can be 1 or 2

setBuilding(self, bldng)
setBuilding Could move into new message

setZone(self,num)
setZone num must be 1 or 2, could move into new/create message

solarGain(self,month)
solarGain Bredem-8 A.4.3 p9-10 in Watts

spaceHeatingReqt(self,month)
spaceHeatingReqt in watts. Bredem-8 A.4.4.8 p14 SM makes changes

specificHeatLoss(self)
specificHeatLoss rate Bredem-8 A.4 p7

tempWithoutHeat(self,month)
tempWithoutHeat Bredem-8 A.4.4.6 p12-13

timeToCool(self)
timeToCool Bredem-8 A.4.4.6 p12

ventilationLoss(self,ach)
ventilationLoss Bredem-8 p7

volume(self)
volume of all the rooms in a Bredem Zone in cubic metres

Site

Source file: SITE.CLS

Inherits from: Object Room

Inherited by: (no descendants)

special kind of room which represents the space outside the building

Class variables: (none)

Instance variables:

id	(From class Room)
name	(From class Room)
lights	(From class Room)
occupants	(From class Room)
equipment	(From class Room)
roomLinkers	(From class Room)
rooms	(From class Room)
casualGain	(From class Room)
regionNum	Ordinal value of UK region
climate	climate object
groundReflectance	"N" or "S"
hemisphere	"N" or "S"
latitude	
longitude	

Class methods:

defaultNew(self)
set up a default site object

Object methods:

climate(self)
return climate

exposure(self)
site exposure, sheltered, normal or severe

findRegionNum(self, name)
Given the region name, find the number

fromIniFile(self, str)
read from database

groundReflectance(self)
groundReflectance

hemisphere(self)
Comment

init(self)
initialise site object

latitude(self)
latitude

longitude(self)
longitude

meanHorizSolarRadiation(self, month)
meanHorizSolarRadiation Bredem-8 A.4.3, p9 and Table 4

meanMonthlyTemp(self, month)
meanMonthlyTemp Bredem-8 Table 10

meanSolarRadiation(self, orientation, month)
meanSolarRadiation Bredem-8 A.4.3.3, p10 and Table 4

outlook(self)
outlook Can be open, superb, or city

region(self)
region

regionNumber(self)
 regionNumber
setClimate(self,aCli)
 set a new climate
setGroundReflectance(self,num)
 setGroundReflectance
setHemisphere(self,str)
 north or south
setLatitude(self,num)
 setLatitude
setLongitude(self,num)
 setLongitude
setRegionNumber(self,num)
 setRegionNumber
shadeFactor(self)
 shadeFactor Bredem-8 A.4.3.4 p10
solarDeclination(self,month)
 solarDeclination Bredem-8 A.4.3, p9-10 & Table 6
sunPath(self)
 return sun path

Element

Source file: ELEMENT.CLS
Inherits from: Object
Inherited by: Floor GroundFloorSolid GFSuspended Opening
 DoorOpening WindowOpening Roof Wall

SRL binding class for windows walls floors etc

Class variables: (none)

Instance variables:

id	for data mapping purpose. SM
name	name
room1	the room adjacent to finish 1
geometry	geometry of the element
finish1	finish 1
constructionType	construction
finish2	finish 2
room2	the room adjacent to finish 2
openings	a collection of openings

Class methods:

defaultNew(self,rooma,ct,roomb)
 initialise a default element
functions(self)
 return all functions
readInVariables(self,obj,reader);
 data mapping

Object methods:

- addOpening(self,opening)**
add a new thing into openings
- admittance(self)**
calculating the admittance
- admittanceLead(self)**
the lead on the admittance in hours
- admittanceMatrix(self)**
calculating the admittance matrix of a construction type
- anyRoom(self)**
return either room1 or room2 if defined in that order of preference
- avgWeightedReflectance(self,room)**
calculates the area weighted reflectance for CIBSE
- azimuth(self)**
azimuth of the element
- bpeMethods(self)**
specifies useful BPE methods, with no parameters
- connected(self)**
return a collection of all elements which are connected to this one
- constructionType(self)**
constructionType
- create(self,r1,f1,ct,f2,r2)**
create an element any arg can be nil but order must be maintained so that we know how to process the construction type
- decrementFactor(self)**
decrementFactor
- decrementLag(self)**
the lag on the decrement factor in hours FP Modified obn 23 Oct 92
- defaultWindowOpening(self,ratio)**
return a default window opening. Remember the Element could be a wall, roof or floor
- diffuseIrradiance(self, hour, sunPath)**
put sunPath as argument for efficiency.
- directIrradiance(self, hour,sunPath)**
put sunPath as argument for efficiency.
- editPicture(self, boundsRect)**
get the editPicture. editPicture for an Element is defined as a vertical cross section of the construction-type, hence walls, windows etc need no further definition. Floors and roofs need further definition
- exportGeometry(self)**
exportGeometry
- externalSpace(self)**
return the external space, or nil if none
- fabricLoss(self)**
return the fabric loss due to the element (mod. LPR 15/1/92 for internal and external elements)
- fabricLoss(self)**
return the fabric loss due to the element
- finish(self,room)**
SRL returns the finish bordering room

finish1(self)
finish1

finish2(self)
finish2

generalType(self)
generalType

geometry(self)
geometry

getInfo(self)
return info about Construction type

graphic(self)
Comment

groundReflectedIrradiance(self, hour, sunPath)
calculates the ground reflected irradiance on an element by adding the parts due to direct and diffuse radiation on the ground. put sunPath as argument for efficiency.

hourlySolAirTemp(self, hour)
hourlySolAirTemp

hourlySolAirTempSwing(self, hour)
hourlySolAirTempSwing

hourlyStructuralGainSwing(self, hour)
hourlyStructuralGainSwing

id(self)
for data mapping

inclination(self)
angle from vertical

information(self)
information about the element

internalAdmittance(self)
internal admittance

internalAdmittanceLead(self)
lead on internal admittance in hours

internalResistance(self)
return the surface resistance of first internal room found

internalSpace(self)
return the (first) internal space, or nil if none. Normally used where only one surface is internal

internalSurfaceFactor(self)
assumes finish1 is internal!

internalSurfaceFactorLag(self)
the lag on the internal surface factor in hours

isDoor(self)
comment

isExternal(self)
returns true if an element is connected to external site

isExternalFinish(self, finish)
returns true if the finish is bordering site

isExternalSpace(self)
returns true if an element is connected to external site

isFloor(self)
isFloor

isInternal(self)
return true if an element is connected only to internal spaces

isInternalFinish(self,finish)
return true if the finish is bordering site

isRoof(self)
isRoof

isVertical(self)
isVertical

isWall(self)
isWall

isWindow(self)
comment

layerFacing(self,room)
return the construction layer facing room

linkTo(self,room)
adds a room to an element. Room1 is set to the link if it has not been defined else room 2 becomes the link. nil is return ?? if element is fully linked

materialFacing(self,room)
returns the construction layer facing room

maximumX(self)
get the maximum x value of the element in the CAD draw.

maximumY(self)
get the maximum y value of the element in the CAD draw.

meanSolAirTemp(self)
calculate mean sola-air temperature

meanSolarGain(self)
meanSolarGain

meanSolarIrradiance(self)
meanSolarIrradiance

minimumX(self)
get the minimum x value of the element in the CAD draw.

minimumY(self)
get the minimum y value of the element in the CAD draw.

name(self)
return name of the element

netSurfaceArea(self)
returns the surface area of this element, subtracting the area of openings

notStorable(self)
Comment

openings(self)
return all windows in an element.

peakHourSolarIrradiance(self)
peakHourSolarIrradiance

peakSolarIrradiance(self)
peakSolarIrradiance

printOn(self, strm)
show type of construction

realAdmittance(self)
getting the real admittance

realDecrement(self)
getting the real decrement factor

realDecrementFactor(self)
getting the real decrement factor

realInternalAdmittance(self)
real internal admittance

realInternalSurfaceFactor(self)
getting the real internal surface factor

realSurfaceFactor(self)
getting the real surface factor

reqUvalue(self)
reqUvalue

resistance(self)
Calculate the total resistance of a construction including surface resistance.

room(self, finish)
returns the room bordering on a finish

room1(self)
room1

room2(self)
room2

setConstruction(self, newConstruc)
sets constructionType

setFinish1(self, fin)
setting finish 1

setFinish2(self, fin)
setting finish 2

setGeometry(self, nodes)
changed to store geometry as a collection of nodes

setGraphic(self, pic)
Comment

setId(self, int)
for data mapping

setName(self, str)
give element a name

setRoom1(self, rm)
setRoom1

setRoom2(self, rm)
setRoom2

setupElement(self, polygon, blg)
setupElement

solarIrradiance(self, hour)
solarIrradiance

specificHeatLoss(self)
returns the steady state conductive heat loss per degree temperature difference through the element

storeDefinitionOn(self, stream, table)
for data mapping

surfaceArea(self)
SRL returns the surface area of this element. Including openings

surfaceAreaLPR(self)
this gives the gross surface area of the element, assuming the geometry has been set up correctly as an ordered polygon) Factor of 1600 is due to an anachronistic use of Imperial units !?

surfaceFactor(self)
complex surface factor

surfaceFactorLag(self)
the lag on the surface factor in hours

surfaceResistance(self,room)
returns the surface emissivity at either room

totalResistance(self)
returns area-weighted resistance. Assumes all data is properly set up including finishes

totalResistanceLPR(self)
totalResistanceLPR

transmittance(self)
returns wall transmittance i.e. excluding internal and external surface resistances

updateGeometry(self,polygon)
This method handles the change of size of the element polygon. But the vertices number does not change.

updateOpeningGeometry(self)
updateOpeningGeometry

updateXYCoords(self,delta)
Updates XY coords

uValue(self)
Calculate the u-value of an element.

vertices(self)
returns a vertex collection for this element

windows(self)
returns all windows in an element

Floor

Source file:	FLOOR.CLS
Inherits from:	Object Element
Inherited by:	GroundFloorSolid GFSuspended
	Formal class for all floors.
Class variables:	(none)
Instance variables:	
id	(From class Element)
name	(From class Element)
room1	(From class Element)

geometry	(From class Element)
finish1	(From class Element)
constructionType	(From class Element)
finish2	(From class Element)
room2	(From class Element)
openings	(From class Element)

Class methods:

defaultNew(self,rooma,ct,roomb)
set up a default floor

Object methods:

editPicture(self, boundsRect)
Get the editPicture

meanSolarGain(self)
return 0

Opening

Source file:	OPENING.CLS
Inherits from:	Object Element
Inherited by:	DoorOpening WindowOpening
	class comment

Class variables: (none)

Instance variables:

id	(From class Element)
name	(From class Element)
room1	(From class Element)
geometry	(From class Element)
finish1	(From class Element)
constructionType	(From class Element)
finish2	(From class Element)
room2	(From class Element)
openings	(From class Element)

Class methods: (none)

Object methods:

generalType(self)
generalType

getConstraint(self)
for drawing in the CAD

getPlanView(self)
returns an ObjectGraphics Plan view of an opening

meanSolarGain(self)
meanSolarGain

surfaceArea(self)
returns the surface area of this opening

updateXYCoords(self,delta)
updates building model coordinates

WindowOpening

Source file: WINDOWOP.CLS
Inherits from: Object Element Opening
Inherited by: (no descendants)

class comment

Class variables: (none)

Instance variables:

id	(From class Element)
name	(From class Element)
room1	(From class Element)
geometry	(From class Element)
finish1	(From class Element)
constructionType	(From class Element)
finish2	(From class Element)
room2	(From class Element)
openings	(From class Element)
dirtCorrectionFactor	for daylight.
visibleSky	for daylight.
asgf	alternating solar gain factor
sgf	alternating solar gain factor

Class methods:

readInVariables(self,obj,reader);
SM

Object methods:

altSolarGainFactor(self)
altSolarGainFactor

azimuth(self)
temp solution.

collectionFactor(self)
collectionFactor Bredem-8 A.4.3.4 p10

dirtCorrectionFactor(self)
comment

finish(self,room)
returns the finish of the glass bordering room

information(self)
information

isWindow(self)
comment

meanSolarRadiation(self, site, month)
meanSolarRadiation Bredem-8 A.4.3.3, p10 and Table 4

netGlazedArea(self)
returns the area of the opening that is glazed or 0 if it is not a window

openBrdmtWinEdit(self)
once a window editor is OKayed this will open a BREADMIT Editor This may not be the right place but is included for testing the dialogue box.

setAltSolarGainFactor(self,num)
setAltSolarGainFactor

```

setDirtCorrectionFactor(self,val)
    setDirtCorrectionFactor

setFrame(self,str)
    str is in a format of "name,area percentage"

setSolarGainFactor(self,num)
    setSolarGainFactor

setVisibleSky(self,num)
    setVisibleSky

solarGainFactor(self)
    solarGainFactor

solarTransmissionFactor(self)
    solarTransmissionFactor Bredem-8 table 8

visibleSky(self)
    return visibleSky

```

DoorOpening

Source file:	DOOROPEN.CLS
Inherits from:	Object Element Opening
Inherited by:	(no descendants)
	Class Comment
Class variables:	(none)
Instance variables:	
id	(From class Element)
name	(From class Element)
room1	(From class Element)
geometry	(From class Element)
finish1	(From class Element)
constructionType	(From class Element)
finish2	(From class Element)
room2	(From class Element)
openings	(From class Element)
Class methods:	
readInVariables(self,obj,reader);	
data mapping	
Object methods:	
isDoor(self)	
return true	

Roof

Source file:	ROOF.CLS
Inherits from:	Object Element
Inherited by:	(no descendants)

General class for a roof. Inherits all the element stuff, however in general the constructionType will be of class PitchedLoftRoof

Class variables: (none)

Instance variables:

id	(From class Element)
name	(From class Element)
room1	(From class Element)
geometry	(From class Element)
finish1	(From class Element)
constructionType	(From class Element)
finish2	(From class Element)
room2	(From class Element)
openings	(From class Element)

Class methods:

defaultNew(self,rooma,roof,ceiling,roomb)
initialise a default roof

Object methods:

admittanceMatrix(self)
calculating the admittance matrix of a construction type

editPicture(self, boundsRect)
Get the editPicture. This should return the slope etc

generalType(self)
generalType

inclination(self)
angle from vertical

reqUValue(self)
returns a u value of self assumes all data is properly set up

storeDefinitionOn(self, stream, table)
data mapping

Wall

Source file: WALL.CLS

Inherits from: Object Element

Inherited by: (no descendants)

Supports multi-layer walls.

Class variables: (none)

Instance variables:

id	(From class Element)
name	(From class Element)
room1	(From class Element)
geometry	(From class Element)
finish1	(From class Element)
constructionType	(From class Element)
finish2	(From class Element)
room2	(From class Element)
openings	(From class Element)

Class methods: (none)

Object methods:

generalType(self)
generalType

getConstraint(self)
for drawing

getPlanView(self)
returns an ObjectGraphics Plan view, modified to add graphical representations of an opening to a wall

setViewPlan(self)
sets the current view to a plan

updateHorizontalWall(self,xory,cs)
updateHorizontalWall

updateOpeningGeometry(self)
Only for rectangle rooms.

updateVerticalWall(self,xory,cs)
updateVerticalWall

NUWindow

Source file: NUWINDOW.CLS

Inherits from: Object

Inherited by: (no descendants)

class comment

Instance variables:

name
frame
framePercentage
glazingType
glazing

Class variables: (none)

Class methods:

defaultNew(self)
sets up a default window

readInVariables(self,obj,reader);
data mapping

Object methods:

diffuseTransmittance(self)
comment

frame(self)
reports on window frame

framePercentage(self)
returns the glazed area percentage

getFrame(self,str)
sets up the glass material

getGlazing(self,str)
sets up the glass material

glazedPercentage(self)
returns the glazed area percentage

glazing(self)
comment

glazingType(self)
reports on window glazingType

notStorable(self)
Comment

printOn(self,strm)
shows type of construction. SM shows the frame type as well as glazing type.

setFrame(self,str)
setFrame

setFramePercentage(self,num)
setFramePercentage

setGlazing(self,gl)
setGlazing

setGlazingType(self,gl)
setGlazingType

solarTransmissionFactor(self)
solarTransmissionFactor Bredem-8 table 8

totalResistance(self)
returns the resistance of glass

ConstructionType

Source file: CONSTRUC.CLS

Inherits from: Object Collection IndexedCollection Array
OrderedCollection

Inherited by: (no descendants)

Construction types for Combine. construc.ini is the file where the data is found

Instance variables:

firstElement	(From class OrderedCollection)
lastElement	(From class OrderedCollection)
name	

Class variables:

\$dtp1ID

Class methods:

defaultCeiling(self)
sets up a ceiling

defaultDoor(self)
sets up a default door

defaultExternalWall(self)
sets up a default external cavity wall

defaultGroundFloor(self)
sets up a default ground floor

defaultInternalWall(self)
sets up a default internal 100mm wall of blockwork

defaultRoof(self)
sets up a default roof ie the ceiling on the last floor

defaultSolidGroundFloor(self)
sets up a default ground floor

defaultSuspendedFloor(self)
sets up a default intermediate floor ie one which is a ceiling to the room below and a floor to the room above

defaultWindow(self)
sets up a default window

getDTP1ID(self)
data mapping

getLayers(self,name)
returns a collection of named layers using name, e.g. name is defaultExternalWall

read(self, picFile)
Read the ConstructionType from picFile

readInVariables(self,obj,reader);
data mapping

resetDTP1ID(self,num)
data mapping

Object methods:

admittanceMatrix(self)
calculating the admittance matrix for a construction type

conductivity(self,idx)
returns the conductivity of the layer at idx

create(self, nm, keys)
create; eg name=DefaultExternalWall, key=ExternalWallLeaves...etc, getPrivateProfileString = "0, 102"...etc which is the code for an external leaf and its thickness (mm)

editHorizPicture(self, boundRect)
Gets a horizontal Picture of Construction, for use when editing floor constructions. Need to reverse order to get the right way up. Use minimum layer display thickness of 15mm for visibility

editPicture(self, boundRect)
Gets a Picture of Construction, for use when editing

editVertPicture(self, boundRect)
Gets a Picture of Construction, for use when editing. Use minimum layer display thickness of 15mm for visibility

getInfo(self)
getInfo

getNameFromSTEP(self,str)
Comment

name(self)
Returns constructionType name

printOn(self, strm)
print name instead of layers

resistance(self, idx)
returns the resistance of the layer at idx

reverse(self)
inverting the order of the elements of a construction. Not sure about this, DJS LPR 15/1/92

setName(self, nm)
Set constructionType name

storeDefinitionOn(self, stream, table)
data mapping

thickness(self, idx)
returns the thickness of the layer at index idx

totalResistance(self)
returns the total Resistance of the construction.

totalThickness(self)
returns the total thickness of the construction

write(self, stream)
Write self to stream

Layer

Source file: LAYER.CLS

Inherits from: Object

Inherited by: TranslucentLayer

Layers are used to make up Elements

Class variables:

\$dtp1ID

Instance variables:

id	used for data mapping
name	name of a layer
thickness	thickness
specificHeat	number
density	number
conductivity	number
vapourResistivity	number
madeOf	material

Class methods:

read(self, stream)
Read the Material from stream

readInVariables(self, obj, reader);
data mapping

Object methods:

admittanceMatrix(self)
calculates the admittanceMatrix of a layer

asString(self)
comment

conductivity(self)
conductivity

create(self,nm)
sets up a layer

density(self)
comment

diffusivity(self)
comment

editPattern(self)
Gets a brush pattern for drawing layer

fromIniFile(self,nm, str)
instantiates a layer from a string which represents a material number

getInfo(self)
getInfo

getMaterial(self,type, idx)
gets a material at index idx from a layer type.

getMaterials(self, type)
puts up a list box of suitable materials for a given construction type

getNameFromSTEP(self,str)
data mapping purpose

id(self)
Comment

information(self)
information

madeOf(self)
comment

name(self)
return name

notStorable(self)
Comment

permeability(self)
comment

phi(self)
defining the dimensionless phase constant for a layer

resistance(self)
average resistance. For special layers, see case statements below. Low e surfaces in cavities result in very close to double resistance; error insignificant in U value calcs. Values are linear interpolation of Tables A3.7, and A3.8 for emissivity. Assumes horizontal or upward heat flow.

setConductivity(self, value)
setting the conductivity

setDensity(self, value)
setting the density

setMadeOf(self,made)
set material

setName(self,nm)
sets the name

setSpecificHeat(self, value)
 setting the specific heat

setThickness(self, value)
 setting the thickness

specificHeat(self)
 comment

storeDefinitionOn(self, stream, table)
 data mapping purpose

surfaceEmissivity(self)
 must have a material

thickness(self)
 comment

vapourResistivity(self)
 comment

write(self, stream)
 Write self to stream

TranslucentLayer

Source file: TRANSLUC.CLS

Inherits from: Object Layer

Inherited by: (no descendants)

Additional Layer class to handle Windows Elements etc

Class variables:

\$dtpIID (From class Layer)

Instance variables:

id (From class Layer)
 name (From class Layer)
 thickness (From class Layer)
 specificHeat (From class Layer)
 density (From class Layer)
 conductivity (From class Layer)
 vapourResistivity (From class Layer)
 madeOf (From class Layer)
 extinctionCoeff
 refractiveIndex
 diffuseTransmittance
 diffuseReflectance
 directSolarTrans
 solarReflectance
 solarAbsorption
 totalSolarTransmittance
 shortWaveShading
 longWaveShading
 totalShading

Class methods: (none)

Object methods:

diffuseReflectance(self)
returns the diffuse reflectance

diffuseTransmittance(self)
returns the diffuse reflectance

fromIniFile(self,str)
set up self from a string

setDiffuseTransmittance(self,val)
setDiffuseTransmittance

surfaceResistance(self,elem)
comment

transmittance(self,angle)
transmittance

Material

Source file: MATERIAL.CLS

Inherits from: Object

Inherited by: Finish

general purpose material

Class variables: (none)

Instance variables:

materialName
conductivity
surfaceEmissivity
diffuseReflectance
density
specificHeatCapacity

Class methods:

readInVariables(self,obj,reader);
data mapping

Object methods:

absorptivity(self)
Comment

conductivity(self)
returns conductivity of the material

create(self,nm,cnduc,emis,reflec,dens,shc)
DJS

density(self)
comment

diffuseReflectance(self)
diffuseReflectance

emissivity(self)
Comment

fromIniFile(self,str)
instantiates a material from a string held in a windows INI file

getNameFromSTEP(self,str)
 Comment

heatTransferCoefficient(self)
 Comment

information(self)
 return information

materialName(self)
 returns name of the material

notStorable(self)
 Comment

printOn(self,strm)
 prints name on workspace

setDiffuseReflectance(self,ref)
 setDiffuseReflectance

setMaterialName(self,str)
 setMaterialName

specificHeatCapacity(self)
 specificHeatCapacity

storeDefinitionOn(self, stream, table)
 data mapping

surfaceEmissivity(self)
 surfaceEmissivity

Finish

Source file: FINISH.CLS

Inherits from: Object Material

Inherited by: (no descendants)

a finish of a construction element

Class variables: (none)

Instance variables:

materialName	(From class Material)
conductivity	(From class Material)
surfaceEmissivity	(From class Material)
diffuseReflectance	(From class Material)
density	(From class Material)
specificHeatCapacity	(From class Material)
heatTransferCoefficient	
emissivity	

Class methods:

defaultNew(self)
 initialises on creation

Object methods:

absorptivity(self)
 Comment

admittanceMatrix(self)
The complex admittance matrix for a finish

create(self,nm,em,cd,rf,htc)
creates a new finish. This is used by finish editor in FinEditorDialog.

defaultAdmittanceMatrix(self,num)
defaultAdmittanceMatrix

diffuseReflectance(self)
returns the diffuseReflectance of the finish

emissivity(self)
return emissivity

heatTransferCoefficient(self)
returns the resistance of self

init(self)
initialises value from default class methods

isHighEmmisivity(self)
sets emmisivity high this is default

isLowEmmisivity(self)
sets emmisivity high this is default

materialName(self)
returns the name of the fininsh material

name(self)
standardises name method

setAbsorptivity(self,num)
Comment

setCIBSEHighEmissivity(self)
sets emmisivity high this is default

setCIBSELowEmissivity(self)
sets emissivity high this is default

setHeatTransferCoefficient(self, value)
setting the heat transfer coefficient

surfaceResistance(self,element)
assumes heatTransferCoefficient has been set up

Equipment

Source file: EQUIPMEN.CLS

Inherits from: Object

Inherited by: Cooker

Sorts of equipment which a building may contain, excluding HotWaterSystem, HeatingSystem and VentilationSystem

Instance variables: (none)

Class variables: (none)

Class methods:

annualElecEnergyUse(self,building)
annualElecEnergyUse in GJ/year. Bredem-8 A.6.2 p25 Passed the building to determine floor areas

lightingGain(self,building)

lightingGain defined in Bredem-8 A.4.2, p8

Object methods:

Cooker

Source file: COOKER.CLS

Inherits from: Object Equipment

Inherited by: (no descendants)

DJS Domestic cooker

Instance variables:

fuel

Class variables: (none)

Class methods: (none)

Object methods:

annualEnergyUse(self)

annualEnergyUse in GJ per year. Bredem A.6.1 p25

bredeHeatGain(self)

bredeHeatGain Bredem 8, A.4.2 (c), p8 Amount of energy the cooker contributes to the heat gain in the kitchen

fuel(self)

fuel type

monthlyEnergyUse(self, month)

monthlyEnergyUse in GJ Bredem-8 A.6.1 p25

Climate

Source file: CLIMATE.CLS

Inherits from: Object

Inherited by: (no descendants)

A class containing all the climate data necessary for the admittance procedure LPR 9/91

Instance variables:

date	the date of the simulation
kd	the CIBSE parameter k-diffuse
kD	the CIBSE parameter k-direct
tMax	the maximum temperature in C
tMean	the mean temperature in C
hPeak	the hour of the peak temperature
cloudiness	

Class variables: (none)

Class methods: (none)

Object methods:

cloudiness(self)
Comment

date(self)
Comment

hourlyExternalTemperature(self)
hourlyExternalTemperature

hourlyExternalTemperatureSwing(self, hour)
Calculates Hourly external temperatures Swing

hPeak(self)
peak hour

init(self)
sets variables for 10 day risk in SE England. SM makes change to the value of tMean and hPeak

kD(self)
kD

kd(self)
kd

meanExternalTemperature(self)
meanExternalTemperature

setCloudiness(self,num)
setCloudiness

setDate(self,dat)
set data for the climate

setHPeak(self,num)
setHPeak

setTMax(self,num)
setTMax

setTMean(self,num)

tMax(self)
return tMax

tMean(self)
return tMean

Network

Source file:	NETWORK.CLS
Inherits from:	Object Collection IndexedCollection Array OrderedCollection
Inherited by:	BGMNetwork a general network as an ordered collection of nodes class
Class variables:	(none)
Instance variables:	
firstElement	(From class OrderedCollection)
lastElement	(From class OrderedCollection)
Class methods:	

set(self,siz,node)
a new Network is always given an initial node

Object methods:

addArc(self,fromNode,arc,toNode)
add an arc, checking collection not empty, node which arc is being added from exists. Add node which arc goes to, to Network if not there already. This network allows multiple arcs between nodes but they must have different values.

arcs(self,node)
returns all values of arcs connected to a node

closed(self)
check network is closed, i.e. all arcs go to nodes in self

connected(self,frnod,tonod)
checks to see if frnod and tonod are connected directly or indirectly via network. Always true for a correct network if both in network, but required when removing arcs which may leave nodes no longer unconnected. Note this takes into account directions of nodes, so the connected relationship is not symmetric in an asymmetric network.

findNode(self,val)
return node whose value is argument val

in(self,elem)
returns nil if elem not found

links(self, value)
returns a set of values that are link to specified node

makeSymmetric(self)
make all arcs go in both directions

nodeValue(self)
return all values of nodes as Set

removeArc(self,frnod,rel,tonod)
Remove an arc, checking collection not empty, node which arc is being removed from exists.

values(self)
returns a collection of all the vertices

BGMNetwork

Source file:	BGMNETWO.CLS
Inherits from:	Object Collection IndexedCollection Array OrderedCollection _Network
Inherited by:	(no descendants) a special network for handling Building Geometric Models
Class variables:	(none)
Instance variables:	
firstElement	(From class OrderedCollection)
lastElement	(From class OrderedCollection)
Class methods:	(none)
Object methods:	

addLink(self,from,arc,to)

given two nodes with a bi-directional arc, checks to see if nodes exist if they do, links to them, if not, adds new nodes, from and to are values not nodes

findEqualNode(self,val)

return node whose value is argument val

NetNode

Source file: NETNODE.CLS

Inherits from: Object

Inherited by: (no descendants)

Abstract node for Network class

Instance variables:

arcs
value

Class variables: (none)

Class methods:

set(self,val)
comment

Object methods:

addExcRelation(self,rel,nod)

add arc only if not already in existence

addRelation(self,rel,nod)

does not allow nodes to be connected to themselves

arcs(self)
comment

arcValues(self)
return values of arcs as OrderedCollection

hasRelation(self,rel,nod)
check to see if identical arc is already attached to node, return true if so

hasValues(self)
check to see if all nodes have values returning true if this is so. Note that nil=nil returns true, nil=0 false.

init(self,val)
comment

printOn(self, aStrm)
Prints a NetNode object on the specified stream.

removeRelation(self,rel,nod)
if arc is already attached to node, remove it. Assumes arcs unique

setValue(self,val)
Sets the value of the Node

value(self)
comment

NetArc

Source file: NETARC.CLS
Inherits from: Object
Inherited by: (no descendants)

Arcs are used in networks. They have direction and a value.

Instance variables:

node	which the arc goes to
value	belonging to the arc

Class variables: (none)

Class methods:

set(self, val, nod)
create a new arc

Object methods:

init(self, val, nod)
comment

node(self)
comment

printOn(self, aStrm)
Prints an Association object on the specified stream.

setValue(self, val)
comment

value(self)
comment

Point3D

Source file: POINT3D.CLS
Inherits from: Object Point
Inherited by: (no descendants)

For 3D points. see Actor 3.0 manual, Ch 11 Actor 4.0 stuff added in as well

Instance variables:

x	(From class Point)
y	(From class Point)
z	Third coordinate

Class variables: (none)

Class methods:

build(self, x, y, z)
build

getXRotateMatrix(self, angle)
gets the matrix for rotation around x axis, angle in degrees

getYRotateMatrix(self, angle)
gets the matrix for rotation around y axis, angle in degrees

getZRotateMatrix(self,angle)
coordinate rotation around z axis, angle in degrees

readDXF(self, dxfFile)
Read the Point from dxfFile

readInVariables(self,obj,reader);
data mapping

Object methods:

+(self, arg)
Infix operation. Add the two points together.

-(self, arg)
Infix operation. Subtract the two points.

=(self, aPt)
Infix operation. Equals. Two 3D Points are equal to one another if their x instance variables are equal and their y and z instance variables are equal.

asMatrix(self)
SM

asPoint(self)

distanceFrom(self,aPt)
SRL returns the distance from one point to another

lineTo(self, hdc)
Methods below here are added in Actor 4.0 DJS

middlePoint(self, aPt, range)
middlePoint

printOn(self,aStrm)
printOn

scale(self, factor)
Scale x,y and z coords of point

set(self, xVal, yVal, zVal)
Set both the x, y and z values of receiver.

setXY(self,newPt)
sets x and y coords only

setZ(self, zVal)
Set and return the z value of receiver.

storeDefinitionOn(self, stream, table)
x,y,z value /40 to change from Actor window unit to meters

xRotate(self,angle)
coordinate rotation around x axis, angle in degrees

xRotateMatrix(self,matr1)
coordinate rotation around x axis, using a matrix argument

xy(self)
returns x and y (2D) point, ignores z coordinate

yRotate(self,angle)
coordinate rotation around y axis, angle in degrees

yRotateMatrix(self,matr1)
coordinate rotation around y axis, using a matrix argument

z(self)

Return the z value of the receiver point.

zRotate(self,angle)

coordinate rotation around z axis, angle in degrees

zRotateMatrix(self,matr1)

coordinate rotation around y axis, using a matrix argument

REFERENCES

1. AA, 1988, Passive solar house design handbook, AA School of Architecture, London
2. Ackerman A.F., L.S. Buchwald and F.H. Lewski, 1989, Software inspection: an effective verification process, IEEE Transactions software, May, 1989, pp31-36
3. Aktas A.Z., 1987, Structured analysis and design of information systems, Prentice-Hall International, Inc.
4. Alexander C., 1964, Notes on the synthesis of form, Cambridge, Harvard University Press.
5. Anderson B. R., 1988, Energy assessment for dwellings using BREDEM worksheets, BRE Information Paper, 13/88
6. Anderson B.R. and L.D. Shorrock, 1989, The development of a monthly calculation for BREDEM - provisional specification, Building Research Establishment Note, N78/89
7. Anderson B.R., A.J. Clark, R. Baldwin and N.O. Milbank, 1985, BREDEM - BRE Domestic Energy Model - background, philosophy and description, Published by Building Research Establishment, Garston, Watford, U.K.
8. AR, 1985, The results of RECORD survey: How are firms with computers faring - and what are the nonusers waiting for?, Architectural Record, June 1985, pp37-41
9. AR, 1988, Computers: Systems show prowess and matures with the profession's involvement, Architectural Record, July 1988, pp37-39
10. ART editorial, 1970, Strategies for architectural research, Architectural Research and Teaching, May 1970, pp3-5
11. Augenbroe G and F. Winkelmann, 1991, Integration of simulation into the building design process, unpublished paper, Department of Civil Engineering, Delft University of Technology, The Netherlands
12. Augenbroe G. and L. Laret, 1988, COMBINE pilot study report, Research paper
13. Augenbroe G. and F. Winkelmann, 1989a, Integration of simulation into building design: the need for a joint approach, Draft paper submitted to 1990 ASME International Solar Energy Conference, Miami, FL, April 1-4, 1990
14. Augenbroe G and L. Laret, 1989b, Information package for the DG XII programme, COMBINE report

15. Bazjanac V., 1975, The promises and the disappointments of Computer Aided Design, in Nicholas Negroponte, editor, Computer Aids to Design and Architecture, Mason/Charter, London
16. Bell D., I. Morrey and J. Pugh, 1992, Software engineering, a programming approach, Published by Prentice Hall International Ltd., 1992
17. Björk B., 1987, The integrated use of computer in construction - the Finish experience, Conference proceeding ARECDAO87, pp18-21
18. Björk B., 1989, Basic structure of a proposed building product model, Computer Aided Design, Vol.21 No.2, March 1989, pp71-78
19. Björk B., 1991a, A unified approach for modelling construction information, Paper to be published at Building and Environment
20. Björk B., 1991b, Building product data modelling - experiences of prototype development, Draft paper submitted to Microcomputers in Civil Engineering
21. Bloomfield D., et al, 1988, Improving the design of low energy passive solar houses, Building Research and Practice, No.6 1988, pp378-384
22. Boehm et al., 1978, Characteristics of software quality, Amsterdam, North Holland
23. Boyle S., 1989, More work for less energy, New Scientist, 5th August 1989, pp37-40
24. Bradfield V.J.(edited), 1983, Information for architecture, Published by Butterworth & Co (Publishers) Ltd
25. Brambley M.R., M. S. Addison, and et al, 1988, Advanced energy design and operation technologies research, Pacific Northwest Laboratory research proposal, PNL- 6255, UC-95d
26. BRE, 1988, Exploiting sunshine in house design, draft design handbook
27. BRE, 1992, BRECSU annual review 1991/92
28. Brewer R, I. Cooper and S. Lera, 1986, People, an unresolved problem in the provision of design guidance and tools for low energy building, Proceeding International Climatic Architecture Congress, 1/3 July 1986, Louvain-La-Neuves, pp237-254
29. Brewer R, 1988a, Finding out what users want in technical publications, Building Research and Practice, No.6 1988, pp346-351

30. Brewer R. and C. Snow, 1988b, Technical publications: the design professionals' response, *Building Research and Practice*, No.1, 1988, pp25-29
31. Broadbent G., 1973, *Design in architecture, architecture and the human sciences*, London, New York, John Wiley & Sons Ltd.
32. Broadbent G., 1988, *Design in architecture, architecture and the human sciences*, London, New York, John Wiley & Sons Ltd.
33. Brown A, 1988, The passive solar programme - A review of progress and a suggest approach to future work, *Solar Heating Steering Committee report*, SHSC(88)P169
34. Bryan H., 1986, Daylighting design tools, *Lighting Design and Application*, March 1986, pp49-56
35. Building Regulation 1990 Part L Approved document, Conservation of fuel and power
36. Capron H.L., 1990, *Computers, tools for an information age*, Published by The Benjamings/Cummings Company, Inc. California USA
37. CEC (Commission of the European Communities), 1988, *European passive solar handbook*
38. CIBSE Guide Section A3, *Thermal properties of building structure*, 1986
39. CIBSE Guide Section A8, *Summertime temperatures in buildings*, 1986
40. CIBSE, 1987, *Window design, application manual*, published by The Chartered Institution of Building Services Engineers, London
41. Coons S., 1975, *Reflections beyond SKETCHPAD*, in Nicholas Negroponte, editor, *Compter Aids to Design and Architecture*, Mason/Charter, London
42. Cooper I, 1984, Barriers to the exploitation of daylighting in building design - U.K. experience, *Energy and Buildings*, 6(1984) pp127-132
43. Cooper I, 1989, SERC, information needs and the construction industry, SERC seminar on Information needs in the construction industry, Darwin College, Cambridge, 12 September 1989
44. Courtney R., 1989, Science friction, *Building (Technology Focus)*, November 3, 1989, pp74-75

45. Cox B.J., et al, 1991, Object-oriented programming: an evolutionary approach, 2nd edition, Addison-Wesley Publishing Co.
46. Cross N., 1989, Engineering design methods, Wiley, New York
47. Danner W. F., 1988, A global model for building-project information: Analysis of conceptual structures, U.S. National Bureau of Standards, PB88-201546, April 1988
48. Darke J., 1978, The primary generator and the design process, in New Directions in Environmental Design Research, Proceedings of EDRA 9, edited by W.E. Rogers and W.H. Ittelson, Washington, pp325-337
49. DBMS committee, 1991, Third-generation database system manifesto, Computer Standards & Interfaces, vol.13, 1991, pp41-54
50. Dell'Isola A.J., 1982, Value engineering the the construction industry, Van Nostrand Reinhold, New York
51. Deutsch M.S. and R.R. Willis, 1988, Software quality engineering, Prentice Hall
52. Diepeveen W.J., 1987, Transfer of knowledge to the building practice, from Information Exchange as a Means for Improving the Quality of Building Research, BVN skriftserie 1987(2) Swedish Council for Building Research
53. DoE, 1989, Energy efficiency best practice programme - A guide for participants, Energy Efficiency Office, Department of Energy
54. DoE, 1990, Building Regulations, Part L, 1990 edition
55. Dubois A.M. and et al, 1991, IDM progress report, COMBINE paper, COMB 91-09
56. Dupagne A., 1991, 'CIB Exploratory action No 5604', Research report
57. EAS, 1989, Energy Calculator: The easy way to complete BREDEM worksheets, Energy Advisory Services Ltd., The Old Manor House, The Green, Hanslope, U.K.
58. Eastman C.M., 1991, The evolution of CAD: integrating multiple representations, Building and Environment, vol. 26, No. 1, 1991, pp17-23
59. EDC (Economic Development Council), 1987, IT futures ... IT can work, National Economic Development Office, London

60. EEO, 1989, Energy Efficiency Best Practice Programme, A guide for participants, Energy Efficiency Office, Department of Energy
61. Elder A.J., V. Powell-Smith and P. Pitt, 1989, Guide to the Building Regulations 1985, Butterworth Architecture
62. Enkovaara E., M. Salmi and A. Sarja, 1988, RATAS project, Computer aided design for construction, Published by Building Book Ltd., Helsinki, Finland
63. Estock R.G., 1989, Three object-oriented environments for the desktop, IEEE Software, May 1989, pp100-103
64. Evans B., 1989, In support of thought, knowledge engineering, Architectural Record, Nov. 1990, pp52-55
65. Evans B., J.A. Powell and R. Talbot, 1982, Changing design, John Wiley & Sons Ltd.
66. Fagan M.E., 1976, Design and code inspections to reduce errors in program development, IBM system Journal, vol.15 No.3 1976, pp128-221
67. Fairley R., 1985, Software engineering concepts, McGraw-Hill Book Company
68. Fallon K, 1990, Beyond computers as pencils, Architectural Record, Nov. 1990, pp28-31
69. Fazio P. and K. Gowri, 1989, A knowledge-based system for the selection and design of roof systems, Building Research and Practice, No.5 1989, pp294-298
70. Fenves S. J. et al, 1990, Integrated software environment for building design and construction, Computer-Aided Design, Vol.22 No.1 Jan/Feb. 1990, pp27-36
71. Fisher T. and S. Doubilet, 1986, P/A technics, intelligent computers, Progressive Architecture, June 1986, pp104-113
72. Frost R., 1986, Introduction to knowledge base systems, Published by Collins Professional and Technical books, London, 1986
73. Gielingh W.F., 1989, AEC product-type modelling, or: 'how to use the AEC-model?', TNO-report, August 1989
74. Goodey J. and K. Matthew, 1971, Architects and information, Research Paper 1, IOAAS, University of York

75. Graham I., 1992, Object oriented methods, published by Addison-Wesley Publishing Company
76. Groak Steven, 1992, The idea of building, published by E & FN SPON
77. Gugelot H., 1963, Industrial design in practice, Zeitschrift der Hochschule fur Gestaltung, January 1963, pp3-5
78. Hall B., 1989, Information needs in the construction industry: a view from BRE, SERC seminar on Information needs in the construction industry, Darwin College, Cambridge, 12 September 1989
79. Harrow P., 1991, An introduction to the standard for the exchange of product model data (STEP) and its application to industry, Seminar handout, Engineering Design Centre, University of Newcastle Upon Type
80. Heath T., 1984, Method in architecture, John Wiley & Sons
81. Henderson G and D. Shorrock, 1990, Greenhouse-gas emissions and buildings in the United Kingdom, BRE Information Paper 2/90
82. Hickling A., 1982, Beyond a linear iterative process?, Changing Design, Edited by B. Evans, J.A. Powell, and R.J. Talbot, John Wiley & Sons Ltd., pp 275-293
83. Hofstater D.R., 1985, Metamagical themas - Questing for the essence of mind and pattern, New York, 1985
84. Howard H. C., R. E. Levitt, and et al, 1989, Computer integration: reducing fragmentation in AEC industry, Journal of Computing in Civil Engineering, No.3 1989, pp18-32
85. IEEE, 1983, IEEE Standard glossary of Software Engineering Terminology, IEEE Standard 729-1983
86. Johnson B. G., 1987a, Information in building research - a management approach, BVN report 1987(2), pp68-74
87. Johnson B. G., 1987b, Information supply to building research - aspects on the research activity and its use of information, BVN report 1987(2), pp21-26
88. Jones J.C., 1970, The state of the art in design methods, Emerging methods in Environmental design and planning, Edited by G.T.Moore, MIT Press

89. Jones J.C., 1976, Design Methods, Seeds of human futures, John Wiley & Sons Ltd.
90. Kalay Y.E., 1985, Redefining the role of computers in architecture: from drafting/modelling tools to knowledge-based design assistants, Computer Aided Design, Vol 17, No.7 1985, pp319-328
91. Karlen, I., 1987, The building research process and its information problems, BVN report 1987(3)
92. Kealy L., 1988, Information requirements for architects, Research Report for SOLINFO, University College Dublin
93. Kendall J., 1992, NESSIE version 0.3c, NIAM to EXPRESS schema
94. Kharrufa S., H. Aldabbagh and W. Mahmond, 1988, Developing CAD techniques for preliminary architectural design, Computer Aided Design, Vol 20 No. 10, 1988, pp 581-588
95. Kim W., 1990a, Introduction to object oriented databases, The MIT Press
96. Kim W., 1990b, Object-oriented database - definition and research directions, IEEE Trsnactions on Knowledge and Data Engineering, Vol.2 No.3 September 1990, pp327-341
97. Kolb D. A., 1976, The learning style inventory technical manual, MacBer and Company, Boston, USA
98. Lansdown J. and T. Maver, 1984, CAD in architecture and building, Computer Aided Design, Vol.16 No.3 1984, pp148 154
99. Lansdown J., 1990, Personal programs possible, Architects' Journal, August 22&29 1990, pp46-48
100. Lawson B., 1980, How designers think, The Architectural Press Ltd.
101. Le Corbusier, 1976, Towards a new architecture, Translated by Frederick Etchells, The Architectural Press
102. Leslie M, 1989, Programming without pain, hypercard potential, AJ Architech Supplement, 22 Nov. 1989, pp34-41

103. Lewis P.T., 1988, Development and evaluation of an interactive video information system for energy efficient building design, Research proposal, Welsh School of Architecture, University of Wales
104. Lockley S. R., 1993, Private communication
105. Lockley S., et al, 1991, Prototype specification for design of external building elements, COMBINE research report, University of Newcastle Upon Tyne
106. Lockley S.R., 1987, The application of knowledge based systems to architectural decision making, proceeding of 1987 European Conference on Architecture, Munich, 1987
107. Lockley S.R., M. Sun, and et al, 1992, Construction Design of External Building Elements, Final report of COMBINE DT-1 project, Department of Architecture, University of Newcastle upon Tyne
108. Mackinder M. and H. Marvin, 1982, Design decision making in architectural practice, Research Paper 19, IOAAS, University of York
109. Mackinder M., H. Marvin, 1982, Design decision making in architectural practice, IOASS Research Paper 19
110. Macleod I. A. and M. Y. Rafig, 1988, Integrated computer aided structural design of buildings, Research paper, University of Strathclyde
111. Marvin H., 1985a, Information and experience in architectural design, Research Paper 23, IOAAS, University of York
112. Marvin H., 1985b, Meeting building designers' need for trade information, BRE information paper, 14/85
113. Marvin H., 1985c, Using experience and publications in building design, BRE Information Paper 13/85
114. Maver T., 1970, Appraisal in the building design process, in Emerging Methods in Environment Design and Planning, edited by G.T. Moore, M.I.T. Press, Cambridge, Mass.
115. Maver T., 1989, Technology transfer - marketing CAD, SERC Seminar, Darwin College, Cambridge, September 12, 1989

116. Mcdermott J., 1982, Domain knowledge and the design process, Design Studies, Vol.3 No.1, January 1982, pp31-36
117. Meta Software Corporation, 1987, Design/IDEF, User's Manual, 150 CambridgePark Drive, Combridge, MA, USA
118. Meyer B., 1988, Object-oriented software construction, Prentice Hall
119. Microsoft Corporation, 1991a, Microsoft MS-DOS 5, user's Guide
120. Microsoft Corporation, 1991b, Microsoft Windows user's guide
121. Microsoft, 1990, Microsoft Windows software development kit, tools
122. Mitchell W.J., 1977, cited from (Stevens, 1991)
123. NBS, 1992, The specification system with the answer, Published by NBS Services Ltd.
124. NCL, 1989, DEMON Specification, School of Architecture, Newcastle University
125. NCL, 1992, IBEEM, an Integrated Building Energy and Environmental Model, Research proposal, Department of Architecture, Newcastle University
126. Newborough M., S. D. Probert and P. A. Page, 1991, Energy edutcation in the UK, problems and perspectives, Energy Policy, September 1991, pp659-665
127. Nicholson P., 1987, More to low cost CAD than meets the pocket, Architect's Journal Supplement, November 25, 1987, pp14-25
128. Oey K. H. and E. Passchier, 1988, Complying with practice codes, Batiment International Building research & Practice, No.1, 1988, pp30-36
129. Ohsuga S., 1989, Toward intelligent CAD systems, Computer Aided Design, Vol.21 No.5, June 1989, pp315-337
130. Oxman R. M. and R. E. Oxman, 1991, Formal knowledge in knowledge-based CAD, Building and Environment, Vol 26 No 1 pp35-40
131. Page J.K., 1963, Review of papers presented at the conference, Conference on Design Methods, edited by J.C. Jonesand P.G. Thornley, Pergamon, Oxford
132. Parand F. and A.M. Dubois, 1990, A uniform format for documenting input/output requirements of building performance evaluation tools within the COMBINE project, COMBINE report, COMB-90-002

133. Plokker W., 1991, Remarks about an Express/STEP interface kit for COMBINE, COMBINE report, August 1991
134. Popper K. R., 1968, The logic of scientific discovery, Hutchinson, London
135. Poston R.M., 1986a, Improving software quality and productivity, IEEE Software, May 1986, pp74-75
136. Poston R.M., 1986b, Software reviews and audits: What are they? Are they enough?, IEEE Software, March 1986, pp71-73
137. Powell J.A., 1988, Intelligent design team members require intelligent education, unpublished paper, School of Architecture, Portsmouth Polytechnic, Hampshire
138. Pressman R. S., 1987, Software engineering, 2nd Edition, McGraw-Hill Book Company
139. Radford A. D. and J. S. Gero, 1985, Towards generative expert systems for architectural detailing, Computer Aided Design, Vol.17 No.9 November 1985, pp428-435
140. Ray-Jones A, 1990, RIBA survey of computer usage 1989, RIBA Practice, July 1990, pp2-3
141. Reddy A., 1991, Barriers to improvements in energy efficiency, Energy Policy, December 1991, pp953-961
142. Reizenstein J. E., 1975, Linking social research and design, Journal of Architectural Research, 4/3, December 1975, pp26-38
143. Research digest No.1, Research and development programmes on solar energy applications to buildings, Commission of the European Community, September 1988
144. Reynolds,R. A., 1980, Computer methods for architects, Published by Butterworths & Co. Ltd, 1980
145. RIBA, 1984, Employment and Earnings survey
146. RIBA, 1988, Job administration, RIBA publication, London
147. Richardson A., C. Jackson and W. Sykes, 1990, Taking research seriously, Department of Health Publication, London, 1990
148. Richens P.,1990, MicroCAD software evaluated for construction, Published by Construction Industry Computing Association, Combridge

149. Riddle R. J., 1988a, The passive solar programme-Domestic sector, Solar Heating Steering Committee report, SHSC(88)P176
150. Riddle R. J., 1988b, The house design studies - a progress report, UKAEA internal report, SHSC(88)177
151. Rosenman M., R. Coyne and J. Gero, 1987, Expert systems for design applications, in J. R. Quinlan, editor, Application of expert systems, Addison-Wesley, London
152. Sanders K., 1989, Emerging trends in architectural CAD software, Architectural Record, March 1989, pp130-135
153. Schmitt G., 1988, Microcomputer aided design for architects and designers, Published by John Wiley and Sons, Chichester
154. SERC seminar, 1989, Information needs in the construction industry - The role of SERC-funded researchers, Darwin College, Cambridge, September 12, 1989
155. Shove E., 1991, Putting science into practice: saving energy in buildings, Research proposal, IOAAS, University of York
156. Slotnick D. L., 1989, Computers and application: an introduction to data processing, Lexington, Mass., D. C. Heath
157. Smith D.J. and K.B. Wood, 1987, Engineering quality software, Elsevier Applied Science
158. Sommerville I., 1989, Software engineering, Addison-wesley Publishing Company
159. Spiby P., 1991, Express Manual, IS 10303 part 11, prepared by ISO TC184/SC4/WG5, January 1991
160. Sprague R. H., 1986, A framework for the development of decision support systems, Decision Support Systems, Edited by R.H. Sprague and H.J. Watson, Published by Prentice-hall, 1986
161. Stevens G., 1991, The impacts of computing on architecture, Building and Environment, Vol.26, No.1, 1991, pp3-11
162. Stevens G., 1991, The impacts of computing on architecture, Builing and Environment, Vol. 26 No.1 pp3-11

163. Stoker D. F., 1991, CAD versus practice, Building and Environment, Vol. 26, No. 1, 1991, pp13-15
164. TNO, 1993, CGE data modelling tool kits
165. Turner, J., 1988, A systems approach to the conceptual modelling of buildings, Proceedings, CIB Seminar on Conceptual Modelling of Buildings, Lund
166. Turner J., 1989, The conceptual modelling of a building, part 1, Computer Integrated Construction, Vol.1, Issue 1, 1989, pp12-19
167. Turner K., 1983, Periodical, Information source in architecture, Edited by Bradfield V.J., Butterworth & Co (Publishers) Ltd, pp124-135
168. Udell J, 1992 Windows NT up close, BYTE, Vol.17 No.10 1992, p167
169. University Funding Council, 1992, Research Assessment Exercise, Universities Funding Council Circular Letter 5/92
170. Vanier D., 1987, Integrated computer aided building design, Conference proceeding, ARECDAO87, ITEC, Barcelona
171. Wallance D.R. and R.U. Fujii, 1989, Software verification and validation: an overview, IEEE Transactions software, May 1989, pp10-17
172. Warren B., 1992, IEA 21 SUBTASK B, Provisional Final Report (3rd Draft), Section 1 Introduction, IEA21RN216/92, July 1992
173. Warthen B., 1989, A history of AEC in IGES/PDES/STEP, Computer Integrated Construction, Vol.1 Issue 1 1989, pp4-8
174. Whitewater Group, 1990a, ObjectGraphics user's manual
175. Whitewater Group, 1990b, Whitewater Resource Toolkit user's manual
176. Whitewater Group, 1991, ACTOR programming, Whitewater Group, Inc.
177. Wiltshire T.J. and S.R. Lockley, 1988, Energy efficiency, do we know how to do it? Unpublished paper, school of Architecture, University of Newcastle Upon Tyne
178. Wiltshire T.J., and S.R. Lockley, 1989, The integration of building performance evaluation tools in a design practice environment, Research proposal, Newcastle University

179. Wix J., 1991, a presentation at a STEP seminar, Wix Mclelland ltd.
180. Wright A., et al, 1992, EKS final report, Department of Architecture, University of Newcastle upon Tyne
181. Yoon C.S., 1987, The attitude of architects towards design information, Edinburgh Architecture Research, Vol.14, 1987, pp97-115